# Studies on Real-Time Handling of Time Dependent Data( Dissertation_全文 )

AUTHOR(S):

Shimakawa, Hiromitsu

# STUDIES ON REAL-TIME HANDLING OF TIME DEPENDENT DATA

Hiromitsu SHIMAKAWA

Department of Information Science

Kyoto University

March 1999

# Studies on Real-Time Handling of Time Dependent Data

Hiromitsu Shimakawa

## Abstract

The introduction of computers into plant control systems is the origin of the development of data acquisition and service (DAS) systems which provide various kinds of service using process values acquired from plants.

"Why a DAS system?" The answer is that it works as a uniform interface of various control machines from views of computer applications which want to use process values inside the control machines. Since there are applications of various purposes, a DAS system inherently has to provide flexible service, which needs complicated computation.

The most important task in DAS systems, however, should be the acquisition of process values without any loss. The integration of the service and the real-time acquisition should be an ultimate goal of DAS systems. Several factors make the integration hard. First of all, the integration must be achieved at an acceptable cost on widely-used commercial off-the shelf (COTS) platforms. Second, engineers in the manufacturing industry have expected for a DAS system to detect significant state changes in plant. Some of them must be notified within time restrictions. Others should be scrutinized with a sufficient time to find symptoms implying potential abnormal behaviors. Difficulty in the customization of a DAS system for each application is also a serious problem. Specific programming disciplines are indispensable to construct a real-time DAS system, whereas an engineer who knows each application is quite unfamiliar with the programming disciplines. Although many DAS systems have been put into commercial use, they do not succeed in the integration of the service with real-time acquisition. Most of their development efforts have been dedicated to customizable service functions.

In the studies presented in the thesis, a real-time active DAS middleware is discussed in order to integrate the service with real-time acquisition. The service covers active service to react state changes as well as passive service activated with requests from applications. The *Active*RING model is proposed to construct the middleware. The model incorporates semantics specific to tasks of DAS systems. The incorporation greatly reduces difficulties in the integration. It brings simplicity in the middleware architecture so that it can be implemented on COTS real-time operating systems at a low cost. The middleware is portable, because the implementation needs no modification of the operating system kernels.

The thesis addresses a data modeling cognizant of the time course. The data modeling is applied to a plant operation supporting system using trend recognition, which has been hardly covered in conventional studies in the temporal data and logics.

In the thesis, a formal method and tools are also provided to allow practical DAS systems to be generated by users without knowledge on real-time computing. In the method, engineers who have expertise on programming for real-time systems have developed a small example so as to make it predictable whether the timing consistency in a DAS system can be maintained. Since the example is small, its development and modification are easy. The example is expanded according to requirements specified by engineers who know applications in a specific plant. Since formal rules and tools expand the example, a mechanism to provide the predictability for the timing consistency is correctly transferred to a target DAS system.

It is also worthy of being emphasized that the studies value the usefulness in practical uses of the middleware. For developed DAS systems, the predictability of timing behaviors and real-time reaction abilities are demonstrated by means of experiments. The suggestions by the plant operation supporting system are compared with operations by an expert in an actual plant. The efficiency of the formal method and the tools for the DAS system generation is evaluated though the development of an actual DAS system.

Because of the simplicity, the real-time active DAS middleware is strong enough to cover most kinds of problems to be addressed in DAS systems. The middleware is a key component to construct open distributed real-time control systems.



# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 DAS System for Time Dependent Data

In the latter half of the century, many evolutions of computing technologies have brought a plenty of computers into process control in various plants. The evolutions include advances in the real-time computing and the data engineering. Principal data in process control are process values representing a plant status. Real-time handling of them enables computers to realize task execution for the process control. Since not only the current process values but also the process value histories are important in the process control, process values are handled as time dependent data, combined with time stamps representing their occurrence time.

As more computers get prevalent in plants, the manufacturing industry has demanded more complicated and more rigorous real-time handling technologies of time dependent data. For example, in the supervisory control[3] which is the complex of control, monitoring, and operation, process values must be acquired without any loss to properly compute control commands and immediately react state changes in a plant. The computation of control commands and the reactions for state changes need service of time dependent data consisting of acquired process values along with conversion of them. To integrate the service of time dependent data with acquisition of them, data acquisition and service (DAS) systems are developed for commercial purposes[19][35][63]. They are also studied in academic societies on the real-time computing[8][32] and the temporal databases[60].

The prevalence of personal computers also inspires plant engineers to connecting real-time systems for process control with non real-time systems for information processing. Many functions in DAS systems should be equipped for the connection between the real-time systems and the non real-time systems[3][21][36]. One objective of the connection is to develop a distributed manufacturing management system which uses process values acquired in a plant at a low cost and in a short term. Numerous software resources such as databases, spread sheet tools, and graphical user interface libraries have been evolved and refined on computers for non real-time information processing. The transmission of data between the real-time process control systems and the non real-time information processing systems would greatly reduce cost and time for the development of manufacturing management systems. Another objective is the establishment of enterprise-wide data interoperability among manufacturing departments and others[43]. Demands from sales departments should be reflected to manufacturing plans to make enterprise activities more efficient. Process data acquired during the manufacturing should be analyzed to invent new products in laboratories. How to analyze relationships in huge amount of time dependent data in manufacturing in a vital discussion topic in data warehouses[31], which is one of academic areas.

The above research and development activities for DAS systems, however, leave several problems from the view points of practical usages. DAS systems should integrate sporadic service of time dependent data with acquisition without any loss. The service includes passive service responding requests from outside and active service to react state changes in a plant. Their data modeling has to be strong enough to represent the time course. DAS systems must be realized at an acceptable cost, and customized to each application with few efforts. Existing achievements for DAS systems do not meet all of these requirements.

This thesis proposes to provide real-time active DAS systems as a middleware on commercial off-the-shelf platforms. In this context, the word, middleware[2], means a set of software functions which work on multiple platforms. It also provides tools to make their customization easy. The real-time active DAS middleware approach discussed in this thesis has



- data modeling which can represent the time course,
- mechanisms to guarantee periodic and reactive data handling, and
- tools to customize DAS systems.

The remain parts of this introductory chapter explains the issues to be addressed by this approach, and the comparison with other approaches in terms of the issues.

## 1.2 Issues for Practical Uses

Functions required in DAS systems are studied in the area of real-time computing and databases. A lot of research efforts have been dedicated to develop more sophisticated functions. Each of them, however, addresses a specific issue. They are not strong enough to cover all requirements for practical uses in an organized manner. The following overviews explain the requirements for practical uses of DAS systems and imply reasons why existing methods cannot cover all of them.

### 1.2.1 Integration of Service with Real-Time Acquisition

In any DAS system, the most important task is acquisition of data in plant. Data which occur in plants must be acquired within time constraints so that they may correctly represent current plant status. In other words, real-time acquisition should be guaranteed. DAS systems must support service of data simultaneous with the acquisition. In this thesis, the service is regarded as the combination of data provision and its activation mode, which categorizes the service into two kinds. One is the passive service which is initiated by an external request to a DAS system, and the other is the active service which is triggered by a DAS system itself. As for the passive service, a DAS system works as a server. The passive service includes periodic transmission of the latest process values and retrieval of process values acquired in a time interval specified by clients. On the other hand, as the active service, a DAS system detects specific state changes in a plant, and notifies predefined

clients of the state changes. The notification is usually accompanied with process values relevant to the state changes.

Since applications issuing requests are abundant in variety, many kinds of requests are sent to DAS systems. DAS systems must be equipped with flexible service abilities to cope with the requests. Process values acquired in various periods should be provided with their time stamps synchronized. Applications would require series of process values on various schemata in arbitrary time intervals. On a specific state change, applications may demand process values before and after the change.

Because of unpredictable amount of handled data, service tasks can cause large transient loads. Some of service tasks should be treated as real-time ones. Dynamics in the system load by the sporadic service makes it difficult to guarantee *a priori* that all tasks on a DAS system complete within their time constraints. Integration of the service with the real-time acquisition means that tasks for the service are accomplished according to their time constraints, without damaging real-time acquisition.

### 1.2.2 Facilities to Represent Time Course

In plant monitoring systems, a lot of attentions are likely to be paid to data indicating the current state of an entity in the plant, so that operators may not fail to miss any change. It is not necessarily adequate for operation of all plants. Operations of some plants often depend on the sequence of events. Occasionally, the length of an interval of one event from another is significant. To properly operate them, operators have to monitor state transitions to the present time as well as the current state.

For example, in some chemical manufacturing plants, effects of operations come forth as the time proceeds. The operators should devote their attention to not only the sequence of operations and their effect appearance but also the time course in the sequence of these events. They recognize the state transition of an entity in the time course.

The state transitions in the time course is also important in terms of preventive maintenance. Trends of process values may often imply potential faults and abnormal behaviors. Suppose a heating furnace for steel billets. When it takes a longer time for the furnace to heat a billet to a target temperature in spite of the usual power consumption rate, operators had better pay attentions to the following state, or check the furnace before it runs into a serious state. Recognition of the state transitions in the time course contributes to finding symptoms of possible future faults and abnormal behaviors.

We need a framework to represent a state transition of a monitored entity in the time course. Many of studies in the temporal databases[60] and the temporal logic[30] in the artificial intelligence emphasize the sequence of events. They cannot be applied to a plant where the time course is significant in the state transition.

### 1.2.3 Customizability

It is worth while to emphasize that an implementation of a real-time system must obey specific programming disciplines so that tasks in the system may be guaranteed to finish by their deadlines. The programming disciplines includes the followings;

- the interactions among tasks should be minimized, i.e. each task should be independent as much as possible,
- the worst case execution time of each task has to be known in advance, and
- the blocking time of one task by another must be bounded.

They must be embodied in the implementation of a real-time system according to constraints in each application. The embodiment is one of main reasons why real-time DAS systems are inherently hard to be implemented. It is almost impossible for people unfamiliar with the disciplines to implement real-time DAS systems.

In spite of the difficulties in the implementation, we have to prepare a customized DAS system which reflects requirements in each plant. There are many kinds of plants, each of which has specific applications. Schemata for data acquisition in one plant is different

from those in another. Values which change rapidly should be acquired in a small period, while a long period may be assigned for acquisition of values changing slowly. We should apply aperiodic acquisition to values which change in discrete manners. In addition to the acquisition based on various schemata and timing, the service of process values is required in manners which vary in each application. Schemata for data provision differ from ones for acquisition. When process values acquired in different period are retrieved, their time stamp should be synchronized. The initiation of a periodic service task means the addition of a new real-time task to a DAS system. On a specific state change, process values before and after the change may be retrieved. It causes an extra load whose occurrence time cannot be predicted. The service must be implemented with programming methods compliant with the programming disciplines, so that the service may be integrated with real-time acquisition.

To apply a DAS system to each plant, it must be customized according to requirements specific to applications in the plant. Since each DAS system varies with an application, tools must be provided to build a DAS system instance suitable for each application. To make real-time DAS systems practical, such tools must hide the disciplines to implement real-time systems so that they can be used by people who are not familiar with the disciplines. Almost all field engineers in each plant are unfamiliar with the disciplines. It is the field engineers that would use the tools.

### 1.2.4 Cost

Another difficulty in implementing practical real-time systems attributes to a trade-off between the logical correctness and the economical factor. It would be expensive to construct a real-time system independent from applications. Let us consider that sporadic tasks have rigorous deadlines for the service of time dependent data. Some of the sporadic tasks share data with other tasks. The maintenance of the logical consistency of the data needs an arbitration mechanism. The maintenance of the logical consistency might be an obstacle to guarantee the completion of tasks by their deadlines. Many research efforts are dedicated to give a guarantee for task completion in a real-time system by deadlines without assuming



the usage of the system. These research efforts, however, result in expensive solutions.

For example, to assure that all sporadic tasks finish by their deadlines, Stankovic and Ramamritham have developed a dynamic scheduling algorithm which assigns tasks, taking resource sharing into account[57]. It needs a scheduling processor apart from processors to execute tasks. Studies to give a static guarantee bring forth the Sporadic server[56], which needs modification of operating system kernels. Either special hardware architectures or special kernel supports are indispensable to realize the real-time handling of sporadic tasks, without any relaxation of constraints. Once real-time systems assume either of them, they are no longer portable; extra efforts are forced when the real-time systems are ported on other platforms.

The manufacturing industry is eager for DAS systems which are inexpensive and portable. From this point of view, it is attractive to realize DAS systems with commercial off-the-shelf(COTS) technologies[61]. It is obvious that COTS technologies lead to inexpensive realization of DAS systems. Such DAS systems are portable because COTS technologies are available on multiple platforms. Of course, there are disadvantages for using COTS technologies. DAS systems realized with COTS technologies are not so strong to satisfy all real-time requirements.

A real-time DAS system should meet all requirements which applications put forth, while it must be inexpensive and portable. As it is the case in many other software systems, a promising way to break through the dilemma is the incorporation of semantics specific to the problem domain. If we neglect to cope with situations which seldom happen in DAS systems, the implementation is substantially simplified. The simplification make it feasible to meet all requirements with COTS technologies. Since the incorporation of semantics leads to proper relaxation of constraints, we can get a good balance between the satisfaction of requirements and the cost. The question is what kinds of constraints should be relaxed for DAS systems.

Table 1.1: Comparison of Approaches

| Approaches | | Real-time works | | Temporal relationship in data modeling | Customization without expertise | cost |
|---|---|---|---|---|---|---|
| | | Periodic works | Reactive works | | | |
| Commercial DAS | | No guarantee | No guarantee | Time stamp | Main interest | Main interest |
| Real-time computing | | Main interest | Main interest | Out of scope | Tools for experts | Special platform |
| DB | Real-Time database | Main interest | Main interest | Out of scope | No practical tool | Special platform |
| | Active database | No Guarantee | Main interest | Out of scope | Out of scope | COTS DB technologies |
| | Temporal database | Out of scope | Out of scope | Order of events | Out of scope | COTS DB technologies |
| Artificial intelligence | | No guarantee | No guarantee | Order of event | Special tools | Out of scope |
| Real-time active DAS middleware | | Guarantee | Guarantee | Time stamp and time course | Special tools are supported | Middleware on COTS platforms |

## 1.3 Conventional Approaches

The requirements in actual uses of DAS systems are related to works in research areas in the real-time computing, the databases, and the artificial intelligence, as well as the technology development in commercial DAS systems. Works in the databases intensively discuss the requirements from the view points of real-time databases, active databases, and temporal databases. In this section, the objective of each conventional approach will be explained to clarify its difference from that of the real-time active DAS middleware approach.

### 1.3.1 Approaches in Commercial DAS Systems

The main objective in existing commercial DAS systems[19][35][36][63] is to provide a customizable DAS system in a low cost, rather than a real-time system. Sophisticated tools are prepared to enable users to customize a DAS system to their applications. Almost all commercial DAS systems are implemented on COTS platforms based on a common hardware architecture and a standard operating system.



Although the commercial DAS systems are equipped with functions for both of data acquisition and service including the passive one and the active one, they do not provide any means to give a guarantee for real-time data handling. The commercial DAS systems would work without any problem until the system load reaches to a certain point. Their behaviors, however, are quite unpredictable in a highly loaded situation. What makes the matter serious is that we cannot know the maximum system load in which they can work correctly. In a DAS system, a burst load can be added at any time. It implies that a DAS system which has worked without any problem suddenly begins to take unexpected behaviors.

From the view point of the data modeling, almost all DAS systems treat each acquired datum as a temporal datum with attachment of the time stamp. Although it is useful for applications cognizant of the time course, DAS systems seldom support to recognize state transitions in the time course. Extra tools other than DAS systems are expected to support it.

### 1.3.2 Approaches in Real-Time Computing

Academic people in the real-time computing society have devoted themselves to provide general supports for real-time computing. The rate monotonic analysis, which is explained briefly in section 2.3.1, is a static scheduling analysis method to assure periodic behavior. The real-time active DAS middleware approach founds its guarantee for periodic tasks on the method. To process active tasks arriving aperiodically by their deadlines, the sporadic server[56] is proposed. Although it can integrate active tasks with periodic tasks, precise CPU time consumed by processing of active tasks is necessary to apply the sporadic server. It needs modification of COTS operating system kernels.

There are several achievements for customization of real-time systems. Adaptation of periodic real-time systems is proposed in [25], where a period of each task is adjusted to satisfy

system-wide schedulability. To make a periodic task set schedulable, [13] proposes a method to modify source codes, maintaining the program dependency. These works emphasize to deal with the modification on the timing behavior such as acquisition period. In an actual DAS system, schemata for acquisition and provision are more likely to be changed than the timing behavior. Apart from the customization of real-time systems, language supports specific to real-time computing are proposed in [20] and [34] for the construction of real-time systems. These supports assume the knowledge on the programming disciplines specific to the real-time computing. Field engineers who are not necessarily familiar with the programming disciplines would customize a DAS system to each application. The supports are not suitable for the DAS system users.

Some works in the real-time computing adopt DAS systems as their targets. A DAS system discussed in [32] is constructed by means of reflecting varieties of timing consistency constraints for data acquisition and service. Although it is noteworthy that the work distinguished timing consistency constraints for the service from those for the acquisition, a delay in the service may lead to a loss of the acquisition, because the system is based on the "producer and consumer" model[27]. The static scheduling analysis method is applied for real-time acquisition in [8]. It discusses neither reactions to specific changes in a plant nor tools for the customization.

### 1.3.3 Approaches with Databases

#### Real-Time Databases

Most of real-time databases historically try to treat all tasks in a uniform manner[14][15], because they seek to improve the ratio for task completion by deadlines without depending any semantics of tasks. Tasks are not always guaranteed to complete by their deadlines. In DAS system with these methods, there could be loss of acquisition during a heavy service task.

By incorporating semantics, Kim and Son[23] categorize transactions into three classes.



Peng and Lin[38] propose a concurrency control method to coordinate acquisition and derivation from acquired data. Although these works deal with consistency in real-time database systems, they have not discussed transient loads caused by active transactions. It is extremely difficult to guarantee all transactions including active ones finish within their time constraints in a method independent from any application. As a few examples, methods proposed in [41] and [54] improve performance of real-time active database systems with dynamic priority assignment. They assume a special support of an operating system, which makes the approaches infeasible on COTS platforms.

As for the customization, ROMPP[64] addresses schema evolution in real-time databases based on performance polymorphism. The work clarifies rules to prevent the schema modification from damaging a guarantee for task completion within time constraints. It, however, has not provided DAS system users with a practical tool. DAS system users are still required to engage in programming activities to modify schemata according to the programming disciplines.

#### Active Databases

In industrial applications, it is important to detect state changes and react them. The concepts of active databases[62] are expected to be useful for industrial applications. The functions for active databases are realized based on the ECA model[29], which executes condition-action rules associated with the occurrence of specific events. The functions can be implemented on COTS database systems.

Since some of state changes in plants need urgent reactions, they must be treated as real-time transactions. As [40] and [54] point out, however, studies of active databases do not deal with real-time transactions. For example, in order that any state change may cause objects to compute a new consistent state, [4] proposes an active database model, where real-time transactions are not considered. Without the consideration of active and real-time transactions, approaches in active databases cannot be applied to DAS systems.

#### Temporal Databases

Temporal databases[9][28][60] emphasize the sequence of events and the temporal relationships of intervals with conventional database technologies. Although it is natural to use temporal data to represent a state transition of a monitored entity, approaches in temporal databases do not address the time course. The temporal databases are not so strong to detect symptoms implying a potential abnormal situation. It cannot be applied to chemical plants, where an effect of an operation appears after a certain time has passed since the operation.

Most of the works on temporal databases pay no attention to processing of real-time transactions. From this aspect, approaches in temporal databases hardly suit for DAS systems. Instead, their interests lie in the maintenance of the consistency of temporal data[55]. [37] surveys research achievements for the consistency in both the acquisition of data item values representing the current plant status and the derivation of new temporal values from other temporal values. The real-time active DAS middleware approach benefits from the research achievements as explained in section 2.4.3.

### 1.3.4 Approach with Artificial Intelligence

The temporal logic in the artificial intelligence gives a vital discussion on temporal data. Like temporal databases, works in the temporal logic emphasize the order of events and the temporal relationships of intervals[30], rather than the time course in the state transition of monitored entities. Dechter proposed a temporal constraint network[7] to maintain the truth in the temporal relationships. To represent future plans, Shoham associates actions with the temporal logic[53]. Neither of them is suitable for representation of the time course, which is indispensable for plant operations, as chapter 5 points out.



## 1.4 Approach for Real-Time Active DAS Middleware

The principal objective of the real-time active DAS middleware approach is the development of practical DAS systems which integrate sporadic service with real-time acquisition. As mentioned in section 1.2, there are two problems to be conquered for achievement of the objective: cost and customization.

To solve the former problem, a real-time active DAS system is realized as a server on top of COTS real-time operating systems, which enable the implementation at an acceptable cost. The modification of the kernel of an operating system should be avoided, because it removes the portability from a DAS system. The real-time active DAS system is a middleware[2], which means on-line functions working on top of multiple platforms. As a solution for the other problem, the approach provides a method to build a real-time active DAS system suitable for each plant without the knowledge on the real-time computing. One of the studies in the thesis addresses off-line tools to generate a real-time active DAS system customized to a specific plant.

The approach has another objective; it realizes a framework to represent the time course. The framework is indispensable for plant operations whose effects appear as the time proceeds and for the preventive maintenance. The framework is hardly ever supported in conventional studies on temporal databases and logics.

The succeeding chapters in the thesis are organized to explain how the real-time active DAS middleware approach addresses the problems and achieves the objectives.

**Integration with COTS Technologies:** Service tasks, especially active ones, bring forth extra loads at unpredictable timing. Because of general-purpose functions supported in COTS platforms, it is impossible to integrate the service with the real-time acquisition in a method independent from any application; the integration needs either a special hardware architecture or an operating system support. In the real-time active DAS middleware approach, some of constraints are relaxed with the consideration of semantics of tasks specific to DAS systems. The tasks specific to DAS systems are

explained in chapter 2 in this thesis, while the semantics of the tasks and the relaxation of constraints will be discussed in the succeeding chapters: a mechanism for DAS systems is proposed to integrate the passive service with the real-time acquisition in chapter 3, and chapter 4 improves the mechanism into a computational model to treat urgent active tasks together with guaranteed real-time tasks.

**Framework for Time Course:** The incorporation of the time course is one of the main differences of the real-time active DAS middleware approach from research achievements in the knowledge and data engineering associated with temporal relationships. The importance of the time course in plant operations is illustrated in chapter 5, which presents a plant operation supporting system using a framework to incorporate the time course.

**Customization of DAS System:** The preparation of customization tools is an essential support for any DAS system. In the real-time active DAS middleware approach, tools are prepared for field engineers to modify schemata and timing behaviors. Chapter 6 covers a formal method to generate a DAS system according to applications. The chapter also explains tools which release users from being conscious of the disciplines to be obeyed for the construction of real-time active DAS systems. The supply of the customization tools has established the real-time active DAS middleware approach.

**Effectiveness:** The approach values the effectiveness in practical uses of DAS systems. To confirm the integration of the service with the real-time acquisition, an experiment is carried out on a prototype in chapter 3. Chapter 4 shows an experimental result to reveal that an instance of the real-time active DAS middleware supports the active service without degrading acquisition abilities. In chapter 5 which evaluates the framework to incorporate the time course, suggestions from the plant operation supporting system during working of an actual plant are compared with operations shown by an expert. The formal method and the tools for the customization in chapter 6 are evaluated through an actual system development.



# Chapter 2

# Constraints and Data in Computing for Process Control

## 2.1 Domain Tasks in Process Control

For development of inexpensive plant systems, COTS software components working on non real-time operating systems should be applied to plants. Therefore, many plants consist of two layers[see figure 2.1]. One is the *real-time system layer*, where completing tasks within time constraints is essential. Each node in the real-time system layer communicates process values within a restricted time using a data highway[44]. The other is the *information system layer*, where the graphical presentation and high-level analysis of process values in a plant are realized by information systems working on non real-time operating systems such as UNIX and Windows. Plant engineers want current and past data acquired in the real-time system layer to be used in the information system layer. To understand phenomena in a plant chronologically, data acquired in the real-time system layer should be stored as temporal data, combined with the time stamp indicating the acquisition time.

Suppose a steel plant where heated billets go through mills one by one. Various tasks are necessary in the plant. Data item values such as the temperature are periodically acquired in heaters, while a photo sensor which detects pass of a milled billet triggers acquisition of the billet status such as the thickness. The exit of a billet from the last heater must be notified to the controller of the first mill to set reference values of the mill pressure. The movement

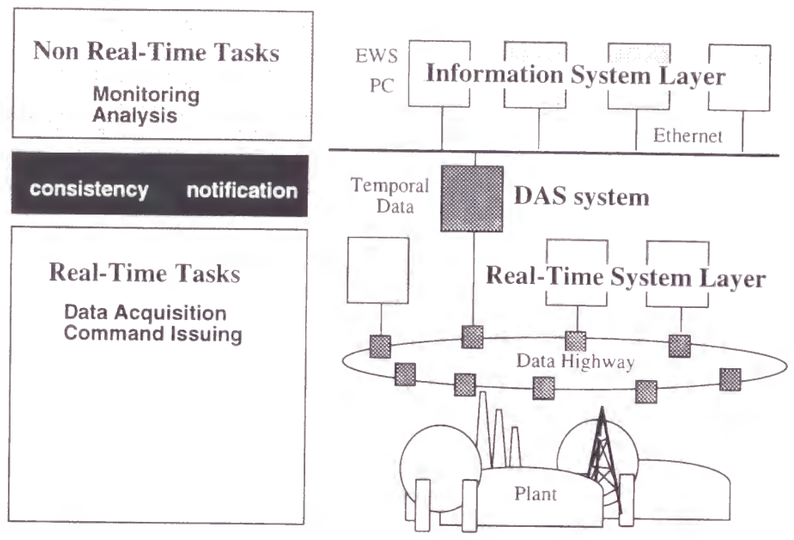



Figure 2.1: Two System Layers in Plant

of each billet must be notified to an information system which displays current position of each billet. In the information system layer, the current feedback values in each process are shown on a graphic display so that operators can monitor them. When operators request, value histories of any process must be retrieved.

A data acquisition and service (DAS) system is necessary to acquire temporal data in the real-time system layer and serve them to clients not only in the real-time system layer but also in the information system layer. It should be equipped with functions shown in Table 2.1.

In the real-time system layer, temporal data can be acquired either periodically or aperiodically. Provision of temporal data simultaneous with acquisition must be supported for both control in the real-time system layer and monitoring in the information system layer. Service is regarded as the combination of data provision and its activation mode. Service is classified into two categories: passive service and active service. The *passive service* is initiated by tasks in clients. The passive service provides temporal data in a requested duration in a requested way. The *active service* detects a specific change in data item values acquired in the real-time system layer, and notifies clients of the change. It triggers setting



Table 2.1: Functions for DAS systems

| | | |
|---|---|---|
| **Acquisition** | **Periodic Acquisition** | |
| | **Aperiodic Acquisition** | |
| **Service** | **Passive Service** | **Periodic Service** |
| | | **On-Demand Service** |
| | **Active Service** | **Real-Time Notification Service** |
| | | **Best-Effort Notification Service** |

of reference values in the real-time system layer and operation guidance in the information system layer. There are two kinds of active service. Real-time notification service achieves urgent reaction such as command issue. Best-effort notification service is prepared to process ordinal reactions without interfering urgent reactions. Table 2.1 uses the word 'notification', instead of 'reaction'. Generally, reaction is a combination of the detection of a state change and the data handling. The data handling consists of the provision of data related to the state change and the calculation of an output with provided data. DAS systems are assumed to perform the detection and the provision. Notification means the combination of the detection and the provision.

## 2.2 Consistency Constraints

Since DAS systems should provide temporal data during real-time acquisition of them, they have many relationships with studies on real-time systems and temporal databases. The timing consistency is the main concern in the former, while the latter value the temporal consistency.

### 2.2.1 Timing Consistency

The timing consistency means that every task in a system finish by its deadline. In studies of real-time systems, three kinds of deadlines are defined from the view point of relationships between the timeliness in task completion and a task value[40][37].

- *Hard deadline tasks* may result in catastrophes if their deadline are missed. When the deadline expires, the task's value can be regarded as a large negative value.
- *Firm deadline tasks* immediately let their values 0, once the deadlines expire.
- *Soft deadline tasks* lessen their values after the deadlines expire. The values are degraded as the time proceeds, and eventually become 0.

Clearly, data acquisition in a plant imparts no value unless changes in the plant are recorded within a specific time. Some service tasks require timely service of temporal data.

### 2.2.2 Temporal Consistency

Two time dimensions, valid time and transaction time are generally discussed in studies of temporal database systems[37]. In this paper, we adopt the following definitions. *Valid time* denotes the time point at which the fact gets true in the real world, while *transaction time* denotes the time point at which the fact is stored in a system. In data acquisition, the difference between the valid time and the transaction time of a data item must be so small that they can be regarded as degenerated[37].

A temporal database system which stores data continuously varying must be temporally consistent as well as logically consistent. While serializability is the main correctness criterion in logical consistency, temporal consistency is based on a validity interval[40]. Temporal consistency has two components.

- *Absolute consistency* means the difference of the valid (or transaction) time of a continuous data item from the current time must be smaller than or equal to the absolute validity interval of the data item.
- *Relative consistency* means data item values needed for the derivation of another data item value must be consistent with their ages[55].



Ramamritham defines temporal consistency[40]. A set of a derived data item and data items used in the derivation form a relative consistency set, which are associated with a relative validity interval. Since all elements of the set have similar age values in [40], the difference between the valid times of any two elements must be smaller than or equal to the relative validity interval. However, in plants, some values are derived from values of different ages. Suppose a furnace whose temperature is sampled every second and which is regulated based on the temperature values acquired in the recent 60 second. Each value of the temperature must be acquired in a short duration which starts from the top of every acquisition period.

We add a tiny modification to the definition. Let $CT$ be the current time. For continuous data item $d$ whose acquisition period is $u$, let $d_{vt}$ and $avi(d)$ denote the valid time and the absolute validity interval of $d$, respectively. Furthermore, assume that $d$ is an element of relative consistency set $R$, and $rvi(R)$ denotes the relative validity interval of $R$. $d$ is correct iff

1. The value of $d$ is logically consistent.
2. $d$ is temporally consistent, that is,

   Absolute consistency:

   $$(CT - d_{vt}) \leq avi(d). \tag{2.1}$$

   Relative consistency:

   $$\forall d' \in R, |d_{vt} - (d'_{vt} + n \cdot u)| \leq rvi(R), \tag{2.2}$$

   where $n$ is an integer.

## 2.3 Guarantee for Real-Time Computing

### 2.3.1 Rate Monotonic Analysis

Many scheduling methods have been studied to maintain the timing consistency. Liu and Layland[26] have presented a sufficient condition to judge whether a given periodic task set is schedulable, *i.e.*, every task in the set finishes within its period.

Suppose periodic and fully preemptable tasks have no interactions with each other. They are scheduled using fix priorities; a task of a higher priority can preempt a task of a lower one at any time. A priority is assumed to be assigned to each task according to the decreasing order of the period. The lowest priority is assigned to tasks which have no period. In other words, those tasks are processed in background. For task $\tau_i (i = 1, 2, ..., n)$, let $T_i$ and $C_i$ are the period and the worst case execution time of $\tau_i$, respectively. Let $p_i \geq p_j$ if $i < j$. It is guaranteed that $\forall \tau_i$ finishes by its period if

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}} - 1). \tag{2.3}$$

The judgment of the schedulability based on the above inequation is referred to as the rate monotonic analysis(RMA)[24]. It is known that the task scheduling based on the RMA is pessimistic. Some of task sets are actually schedularable even if the above condition does not hold.

When some tasks arrive aperiodically, an analysis in the same way can be applied by regarding the periods of the aperiodic tasks as their minimum arrival interval. The analysis is referred to as the deadline monotonic analysis, which is a simple extension of the RMA. It is obvious that the task scheduling with the deadline monotonic analysis is more pessimistic than that with the RMA.

Although the RMA is pessimistic, it is one of a few practical methods for the task scheduling, because the preemptable scheduling based on fixed priorities are supported in most of COTS real-time operating systems. The real-time task scheduling based on the RMA can be implemented without modification of the kernels of existing real-time operating systems.



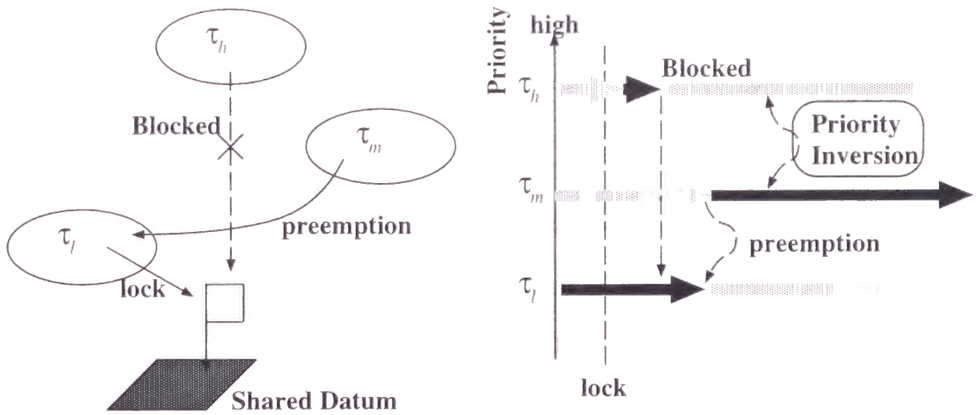



Figure 2.2: Priority Inversion Problem

### 2.3.2 Priority Inversion

Since the RMA assumes that all tasks have no interaction with each other, it must be improved to be applied to actual real-time systems. When a task shares a resource with others, the priority inversion problem occurs.

Suppose that task $\tau_h$ of priority $p_h$ shares a resource with task $\tau_l$ of priority $p_l$ which is lower than $p_h$. When the shared resource stores modifiable data, a mechanism to exclude simultaneous access by multiple tasks are necessary to maintain the logical consistency. While $\tau_l$ updates the content of the shared datum, it should lock the shared datum. $\tau_h$ which tries to access the shared datum later has to wait for $\tau_l$ to unlock the shared datum. Although the lower priority task takes a precedence over the higher priority one, the behavior of each task is predictable if $\tau_h$ and $\tau_l$ which are related with the exclusion are designed so that the blocking time may be bound.

The protocol, however, causes a serious problem when $\tau_m$ locking the shared datum is preempted by task $\tau_m$ of priority $p_m$ which is higher than $p_l$ but lower than $p_h$, as shown in figure 2.2. Since $\tau_m$ preempts $\tau_l$, it also blocks $\tau_h$. In the figure, an arrow of a thick color means that the task is executed, while a pale arrow means the task is not executed. The problem is referred to as a *priority inversion problem*, where a higher priority task

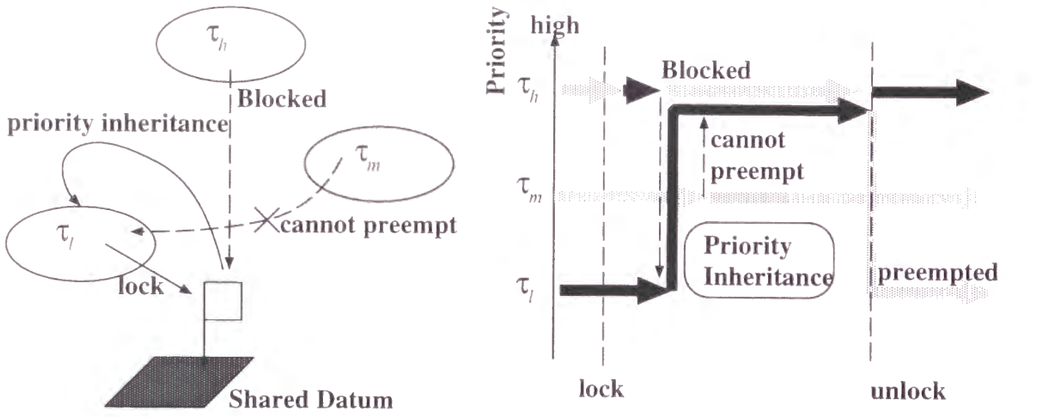



Figure 2.3: Priority Inheritance Protocol

is blocked by a lower priority task with which it has no interaction for an unpredictable time. The blocking time cannot be bound with the execution time of the tasks related with the exclusion. To avoid the priority inversion problem under the conventional exclusion protocol, tasks relevant to the exclusion have to be conscious of the behavior of $\tau_m$ which has no relationships with the share datum. It makes the design of real-time systems too complicated.

### 2.3.3 Priority Inheritance Protocol

The priority inheritance protocol[39] is proposed to solve the priority inversion problem. Suppose task $\tau_h$ of priority $p_h$ tries to access a shared datum which has been locked by task $\tau_l$ of $p_l$, a lower priority than $p_h$. To prevent task $\tau_m$ of a priority which is higher than $p_l$ but lower than $p_h$ from preempting $\tau_l$, the priority inheritance protocol makes $\tau_l$ inherit the priority of $\tau_h$ from the time of the access by $\tau_h$ to the time of the unlock by $\tau_l$, as shown in figure 2.3. While the priority is inherited, $\tau_m$ cannot preempt $\tau_l$. When $\tau_l$ unlocks the shared datum, the priority of $\tau_l$ returns to $p_l$. Since the priority is lower than $p_h$, $\tau_h$ can preempt $\tau_l$. The blocking of a higher priority task by lower priority tasks is caused by tasks related with the exclusion under the priority inheritance protocol.



The time in which a lower priority task blocks a higher priority one can be bound using the time necessary for the former to execute codes during the exclusion. When the worst case blocking time is bound, it has been proved that a task set is schedulable[39] if

$$\sum_{i=1}^{n} \frac{C_i}{T_i} + max(\frac{B_1}{T_1}, \frac{B_2}{T_2}, ..., \frac{B_{n-1}}{T_{n-1}}) \leq n(2^{\frac{1}{n}} - 1), \tag{2.4}$$

where $B_i$ is the worst case blocking time of task $\tau_i$ by a task of a lower priority than $p_i$.

## 2.4 Modeling of Time Dependent Data

### 2.4.1 Data Items

A DAS system handles data items in schemata. Data items are classified with two dimensions.

One dimension is the source of the data item value[37]. The value of a *base data item* is directly acquired from the real world through a sensor, while the value of a *derived data item* is calculated from other data item values.

The other dimension is whether a data item is continuous or discrete[23]. The dimension stresses how a value changes. A *continuous data item* represents the state of an external object which is always changing along with the time. The value of a continuous data item may become invalid over the course of the time. The value of a *discrete data item* does not change until it is explicitly updated.

### 2.4.2 Scenes and Series

Time dependent data used in this paper represent value histories of data items in the real world. A scene and a series[47] are defined as time dependent data to represent value histories.

A *scene* is a set of data item values combined with a time stamp. It represents that the data item values are acquired at the time indicated by its time stamp. Since a scene includes a set of data item values, it is associated with a schema which is specific to each application

In the acquisition, a schema associated with acquired scenes is referred to as an *acquisition schema*, which is an internal schema in a DAS system. When a client requests a DAS system to provide data item values, the client chooses data items relevant to its purpose. A schema used in the provision, which is an external schema, is referred to as a *provision schema*.

A *series* is a chronologically ordered set of scenes. In this paper, a series is represented by the succeeding elements:

- basis $b$ meaning a base time point,
- $n_e$ meaning the number of scenes whose time stamp is earlier than $b$,
- $n_l$ meaning the number of scenes whose time stamp is later than or equal to $b$,

In the acquisition, a scene and its time stamp in a series indicate the acquired values and the acquisition time, respectively. In the provision, what a scene and its time stamp indicate varies with usages of the series. Suppose a series is used for monitoring of a continuous data item value. Monitoring usually requires a value history in a specific rate, i.e., a series including one scene every rate. In the series, a scene and its time stamp show a representative value in one of intervals divided by the rate and the interval associated with the representative value, respectively. The time stamp of each scene should take a value to identify the interval rather than the precise acquisition time. On the contrary, for a value history of a discrete data item, the time stamp should represent the valid time of the data occurrence. The acquisition time is used for the time stamp.

The mode of a series represents the meaning of scenes and time stamps in a provided series. There are three rate-based modes and one non rate-based mode. In rate-based modes, the $i$-th scene in a series indicates the representative value of the $i$-th one of intervals divided by a specific rate. When a series is requested in one of the rate based modes, an allowance is specified to determine which scene indicates the representative value of a specific interval. For a series provided in a rate-based mode, let $b$, $r$, and $a$ be the basis, the rate, and the allowance, respectively. Suppose scene $s$ whose acquisition time is $t$. Scene $s$ can be regarded



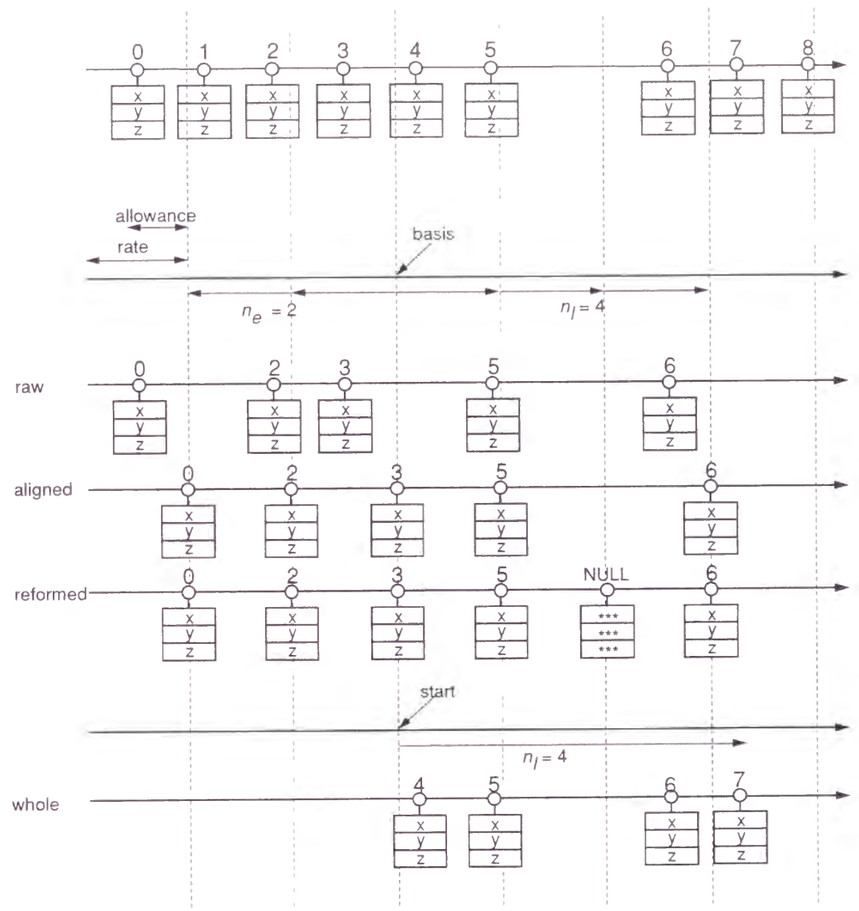



Figure 2.4: Series in Rate-Based Mode

to indicate the representative values in the $i$-th interval, if

$$b + i \cdot r - a < t \leq b + i \cdot r, \tag{2.5}$$

where $i$ is an integer such that $-n_e \leq i \leq n_l - 1$. When $b$ is the current time and $i$ is equal to 0, scene $s$ is the latest acquired scene. As it is shown in the figure 2.4, the three rate-based modes are the following:

**raw mode** where the time stamp shows the acquisition time,

**aligned mode** where the time stamp shows one of intervals divided by $r$, and the series contains no scene if any value is not acquired in the interval.

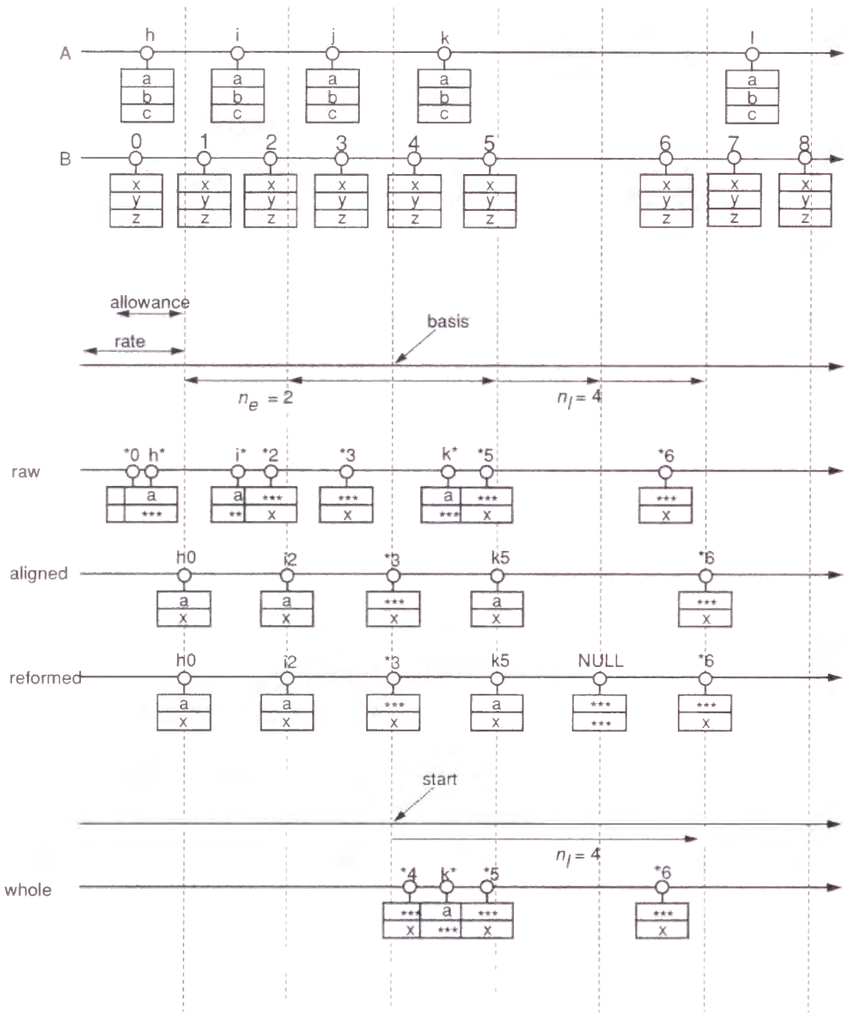



Figure 2.5: Outer Join of Series



**reformed mode** where the time stamp shows one of intervals divided by $r$, and the series contains a null scene if any value is not acquired in the interval,

In the raw mode, the time stamp in the acquisition is used in the provision as it is. The time stamp of each scene is modified to $b+i\cdot r$ in the aligned mode and the reformed mode. A null scene is inserted between the fifth scene and the sixth one in the reformed mode.

A client can request a series which includes all acquired scenes in the duration. In such a case, the series is provided in the non-rate based mode. The whole mode is the non rate-based mode.

**whole mode** where the series contains every acquired scene with its time stamp representing the acquisition time.

The figure depicts the provision of four scenes acquired after the basis in the whole mode.

### 2.4.3 Joining Series

A period appropriate to a continuous data item is adopted when its value is acquired periodically. Multiple series whose rates are different are joined to check relationships among value histories of various continuous data items. The outer join operation[6] is performed with the attribute of time stamps regarded as a key[see figure 2.5].

In the rate-based modes, the scene whose time stamp value is largest is selected for the interval shown in condition (2.5), if multiple scenes are found in the interval. The allowance in condition (2.5) corresponds to the relative validity interval in condition (2.2). Since the time stamp is set to $b+i\cdot r$ in the aligned mode and the reformed mode, original scenes are joined into a single scene for each interval. For the row mode, more than one scenes may be created in each interval. The our join operation sets null values attributes original scenes do not have. Series in the whole mode is handled in the same way as ones in the raw mode, except that all original scenes appear in the joined result.

# Chapter 3

# Guarantee for Firm Real-Time Acquisition and Service

## 3.1 Flexible Service during Real-Time Acquisition

Recent plant systems require functions for information processing such as data visualization and data analysis as well as functions for control. As it is explained in section 2.1, a plant system usually consists of the two layers: the real-time system layer where real-time behavior of each system component must be guaranteed for the controlling, and the information system layer which values flexiblity such as easy (re)configuration of components and data interoperablity among components. Industry is egar for a DAS system as a server to provide process values for both control systems in the real-time system layer and information systems on open operating systems such as UNIX or Windows. The provision of data for the latter is important, because existing information systems greatly contribute to reducing implementation costs of the information processing functions. The DAS system must acquires data in the real-time system layer without any loss and provides the data for not only control systems but also information systems, according to their various requests. Since process value histories are important in the controlling, the DAS system should handle the data as temporal data which indicate process values and their valid time[9] [28][42]

Needless to say, DAS systems must maintain the logical consistency for readingwritting of data managed in it. In addition to that, the DAS systems must maintain the timing

consisntecy that acquisition and provision finish within time constraints[22]. There are various activities which need the process value histories in the real-time system layer and the information system layer. DAS systems should also be equipped with flexible retrieval functions to provide data suitable for each of them. Although a DAS system proposed in [32] for avionics systems provides data during acquisition, it based on the producer and consumer model; it stops acquiring when an internal buffers are full with acquired data because of delay of data provision. Studies on real-time databases focus on real-time handling of requests to update and retrive data aperiodically[15][17][40]. Mothods proposed in these studies are hard to implement on COTS platforms at an acceptable cost. Recent plants are eager to DAS systems which support flexible data service simultaneous with real-time acquisition[21].

This chapter starts with investigating problems to be addressed and tasks to be realized for the connection of the two layers explained section 2.1. Flexible data service according to requests during real-time acquisition is discussed, which suggents an architecture specific to DAS systems.

## 3.2 Connection of Two Layers

### 3.2.1 Requirements

In DAS systems which connect the real-time system layer and the information system layer, we have to integrate passive service and active service with acquisition[see table 2.1]. To make the explanation simple, this chapter discusses the integration of passive service with acquisition[1]. For the integration, the following problems should be addressed.

**Timing consistency:** In DAS systems, the most important task is acquisition of base data item values. Data acquisition is no use unless the value of any base data item is acquired within its absolute validity interval. It is also important for programs in both in the real-time system layer and the information system layer to recognize the current status of the plant. The latest acquired data should be transmitted to the programs

[1]The integration of active service is discussed in the succeeding chapter.



periodically. The timing consistency must be maintained for acquisition and service. To make plant systems dependable, we have to give a guarantee for these tasks to finish by the deadlines in the design phase. In other words, the design of DAS systems must be predictable.

**Flexible provision:** A DAS system is expected to provide data with not only control systems in the real-time system layer, but also applications in the information system layer. Especially, the applications such as data analyzers and graphical monitors request data in formats appropriate to their purposes. Retrieval functions of DAS systems must be flexible. Since the applications in the information system layer generally work on non real-time operating systems, they may sometimes fail to receive data DAS systems provide. Some means should be given for the failure.

**Multiple priorities:** A DAS system is a component in a distributed system where many programs may request data at the same time. Some requests have to be processed urgently, while other do not. In a DAS system, an urgent request which arrives later must be able to take a precedence over those have been processed.

**Portable system:** Any DAS system which needs a special platform is not acceptable for industry which are sensitive for costs. DAS systems must be portable; they should be able to work on multiple platforms, including hardware and operating systems. For example, a DAS system dependent on a specific operating system is not preferable, because upgrades of the operating system may compel the modification of the DAS system. In this sense, the middleware approach[2] is effective to construct a DAS system.

### 3.2.2 Three Kinds of Tasks

DAS systems must provide the flexibility in addition to maintaining the timing consistency. From the view points of these requirements, it is significant to investigate essential tasks in DAS systems. They are classified into the three categories:

1. acquisition of scenes and their management for a long time,

2. periodic service of the latest scene, and

3. on-demand service of a series in an arbitrary duration.

Requirements for maintaining the timing consistency are strongest in the first categories. Data acquisition should be successfully completed by a deadline. The deadline corresponds to absolute validity interval associated with the data item. Data item values must correctly represent current plant status. Deadlines for acquisition must be not only firm but also short. Once process values are acquired within deadlines, temporally correct data can be retrieved using the time stamps. Data item values acquired with the timing consistency enable to maintain relative temporal consistency. Requirements for maintaining the timing consistency in the second category are stronger than those in the third, but weaker than those in the first. DAS systems periodically transmit the latest acquired data to clients in the real-time system layer and the information system layer. Transmission in a shorter period than the acquisition would be useless. Delay in graphical presentation of acquired data to operators is not critical as described in [32]. The third category requires little in maintaining the timing consistency. There are few urgent tasks for the on-demand service. Clients usually request to retrieve past series for the on-demand service, and rarely impose deadlines on retrieval of the series.

On the other hand, requirements for the flexibility get stronger in the order of the categories. We can determine, in advance, all data items to be acquired and almost all data items to be periodically transmitted, as well as their periods. Applications which require the on-demand service, however, need series of various schemata in various rate.

## 3.3 Predictable Design with Circular Areas

It is hard to realize architectures which studies on real-time databases propose on COTS platforms. This paper proposes an architecture specific to realization of tasks essential to



DAS systems for plants, which enables a DAS system to be implemented as a middleware on COTS platforms.

### 3.3.1 Multi Thread Object with Two Ports

A multi thread object[20] is proposed as a paradigm to construct a modularized real-time server. A multi thread object consists of more than one threads and a shared memory between them. It receives a request from a client, and executes methods to accomplish the request.

A server constituted by a single thread is not desirable for real-time applications because it prevents an arrival of a high priority request from preempting calculation for a low priority request which has arrived earlier. A multi thread object solves the problem using two kinds of threads: a chief thread and working threads. The chief thread determines a method to accomplish a request when it arrives. It does not execute the method but entrusts the execution of the method to one of working threads so that it may immediately check whether another request arrives or not. The working thread accomplishes the request by execution of the method. Each working thread is scheduled based on its priority. An urgent request is accomplished by a high priority thread, which may preempt preceding calculation by a lower priority thread, in order to finish the request earlier.

To acquire data exchanged in the real-time system layer interfering control systems as little as possible, the multi thread object should be modified without loosing the functions to accomplish request according to their priorities. As shown in figure 3.1, two kinds of ports are equipped: ones for acquisition from control systems, and the others for communication with clients.

A port for acquisition is implemented with either a memory mapped device which looks like a shared memory with control systems, or a queue to which control systems post data. If a memory mapped device is used, a relationship of a control system to the multi thread object can be regarded to the writer-reader model[27]. On the memory mapped device, control systems periodically write data, which is periodically duplicated into the shared

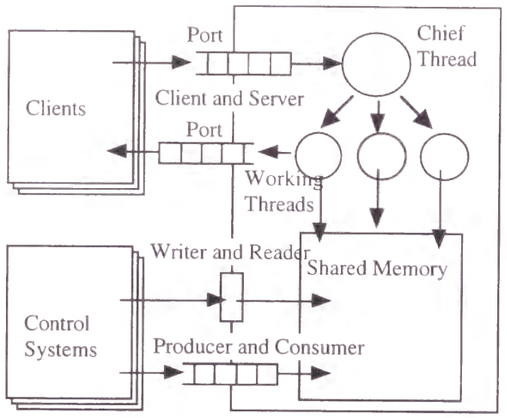



Figure 3.1: Multi Thread Object

memory inside the multi thread object. The multi thread object can acquire data without interfering activities of control systems. If a queue is used, a relationship of a control system to the multi thread object can be regarded to the producer-consumer model[27]. Data are dequeued on the queue by a thread in the multi thread object. The dequeuing task by the thread is scheduled so as to prevent the queue from getting full. What control systems have to be conscious is only posting data to queue. The multi thread object minimizes the interference of activities of control systems.

The multi thread object works as a server for clients. A port for communication with clients is implemented with a queue. When a client send a request to the multi thread object, it waits for a response. The chief thread in the multi thread object gets the request and forward it to a working thread, which sends a response after it accomplishes the request.

### 3.3.2 Two Level Circular Storages

To hold temporal data as scenes for a long time, circular storages are equipped with the multi thread object. As it is pointed in [32] and [45], they constitute two level circular storages: ring buffers in the main memory to hold recent scenes and contiguous disk areas to hold past ones. The contiguous disk areas also configure a circular storage far bigger than a ring buffer.



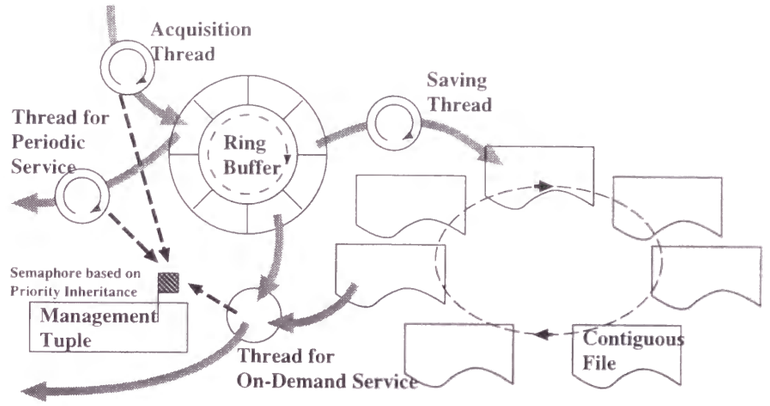



Figure 3.2: Two Level Ring Structure

When a thread in the multi thread object acquires data from a port for acquisition, it forms a scene, by attaching a time stamp indicating the acquisition time to the data. The scene is written on a ring buffer in the main memory. Since scenes are acquired continuously, the latest acquired scene overwrites the oldest scene on the ring buffer. For fear old scenes should disappear, a thread for saving duplicates a bunch of scenes on the ring buffer in one of contiguous files on a disk before they are overwritten, as it is shown in figure 3.2. The two level circular storages allow to cache recent scenes in the main memory and to store many past scenes in disks for a long time.

### 3.3.3 Predictability with Partial Exclusion

Since acquisition tasks and retrieval tasks share circular storages, their access must be arbitrated with a mutual exclusion mechanism. If a retrieval task which make a long access to a circular storage excluded an acquisition task during all access time, the acquisition task may fail to write the latest acquired scene to a ring buffer. To avoid the problem, we should note that only one scene on the ring buffer is overwritten at any time. Each thread to execute a task locks not the whole circular structure but the management tuple, using a semaphore based on the priority inheritance protocol[39]. The management tuple indicates the scene being overwritten[50]. The management tuple holds the time stamp of the latest scene and its position in the ring buffer. It also indicates the last scene which has been duplicated in

a contiguous file, as it is shown in figure 3.2.

When the latest temporal scene are ready to be written into a ring buffer, the thread to write the latest scene locks the management tuple during it accesses the ring buffer. After the thread updates the management tuple, it releases the lock.

For the consistency of scenes to be read from a ring buffer, each thread locks the management tuple. A thread to retrieve scenes from a circular area locks the management tuple before the access. It releases the lock as soon as it finishes reading the management tuple. Since it locks the management tuple only at initial time, it does not interfere threads for tasks of higher priority. There are two kinds of retrieval services: the periodic service and the on-demand service. Using the management tuple, a thread for for the periodic service finds the position of the scene the acquisition thread has just written. A thread for the on-demand service or the best-effort notification service uses the management tuple to calculate which scenes are expected to be stable on the ring buffer at that time. Threads for the retrieval services read the two kinds of scenes from contiguous files: those have been overwritten on ring buffers and those are likely to be overwritten.

The management tuple is so small in size. It is excluded based on the priority inheritance protocol explained section 2.3.3. The blocking time of each thread by threads of lower priorities can be small and bound, which allows the RMA can be applied to give a guarantee that all tasks finish by their deadlines. A DAS system can be constructed so that its timing behavior may be predictable.

### 3.3.4 Reading in Background

There are requests of various priority. A higher priority request which arrives later should take a precedence over lower priority requests which have arrived earlier. A task for retrieval of a low priority may be preempted many times by other tasks. The task gets the contents of the management tuple at initial time. Since it is processed in background, it can be preempted in any time, as far as it makes access to ring buffers. It, however, cannot be preempted during disk access. Retrieval of a long series from a disk is accomplished by



repeated read operations for small size of data on a disk, so that tasks of higher priorities can preempt the retrieval task making access to a disk. Every after the read operation, the retrieval task checks whether tasks of higher priorities wait the disk access. If anyone waits, the retrieval task yields.

A scene read by a retrieval task might be inconsistent, because a thread for acquisition may overwrite the scene. The inconsistency is detected using a time stamp, and compensated with the scene duplicated in a contiguous file. When a scene being read is overwritten, the time stamp of the scene is inconsistent with its expected value. After a retrieval task reads a scene from a ring buffer, it checks the time stamp of the scene. If the time stamp is inconsistent, the retrieval task reads the scene again from a contiguous file.

### 3.3.5 Flexible Provision

There are various clients in a plant system constructed in a distributed way. Some control systems request the periodic service of the latest scene. Applications in the information system layer may need a long series on specific process values to analyze quality of products. It is necessary for provision methods to be described so as to provide time dependent data suitable for client purposes.

A provision method consists of a part to retrieve data based on an acquisition schema from the circular storages and a part to convert retrieved data to data based on a provision schema. With the exclusion mechanism, the logical consistency on the data to be retrieved are maintained in the former part. The latter part is realized with procedures independent from the former part. A precise estimation of the execution time of the procedures enables the RMA to be applied for the maintenance of the timing consistency.

Clients specifies a provision schema in the request to retrieve a series. To make a series to be provided, scenes based on the specified provision schema is retrieved from the circular storages one by one. Some of the specified provision schema are associated with more than one acquisition schemata. The construction of a scene based on the provision schema needs retrieval of all scenes based on the associated acquisition schemata, while each circular

storage corresponds to a single acquisition schema. In a provision method, all scenes based on the associated acquisition schemata are retrieved from corresponding circular storages. The scenes are joined according to the method explained in section 2.4.3. Data items are selected from the joined scene to construct a scene based on the specified schema. A series to be sent for the client is created by the repetition of the above sequence.

## 3.4 Real-Time Data Server

The acrchitecture explaind the previous section enables to design a DAS system, Real-Time Data Server(RTDS)[51]. This chapter explains a prototype of RTDS which realizes

- the acquision inclusing
  - the periodic acquisition, and
  - the aperiodic acquisition,
- the passive service inclusing
  - the periodic service, and
  - the on-demand service.

in the fuctions shown in table 2.1. Figure 3.3 depicts the components of the prototype.

### 3.4.1 Guarantee for Timing Consistency

The protptype is a multi thread object which contains circular storages. Accesses to a circular storage are arbitrated with the managment tuple. Each thread in the prototype has a priority determined by the RMA.

The preseice estimation of the execution time and the blocking time of any task is necessary to apply the RMA. What causes the blocking in the prototype is nothing but the exclusion of the management tuple. As it has been explained, the blocking time is small and bound.



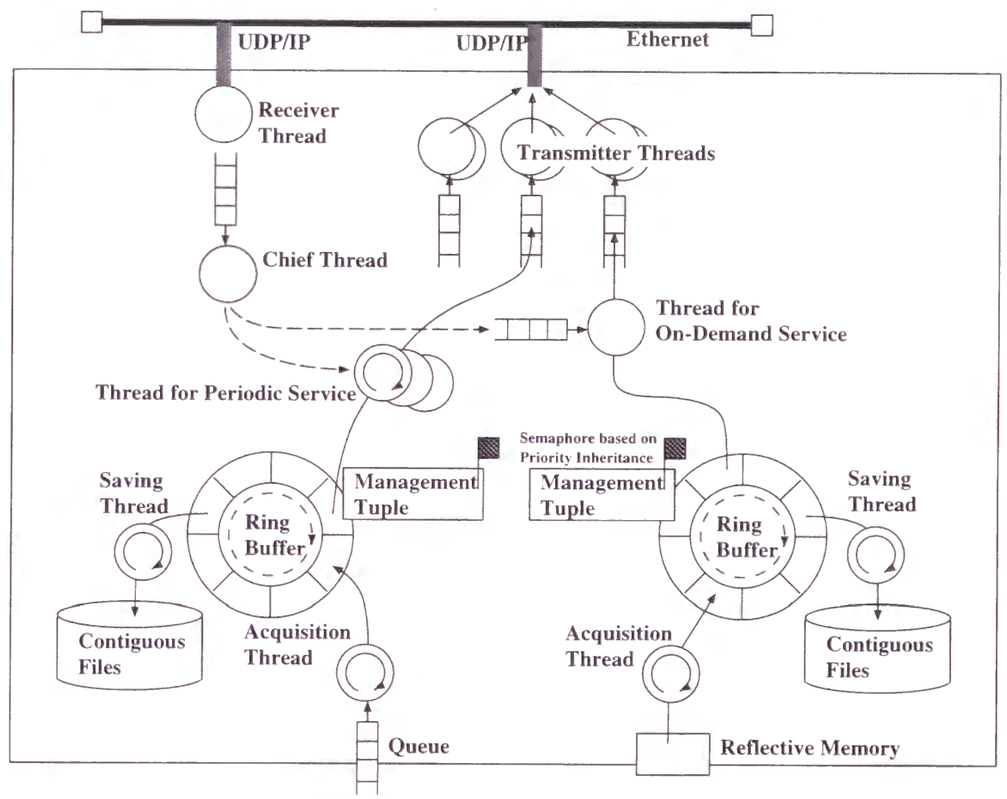



Figure 3.3: Components of RTDS

### 3.4.2 Acquisition and Saving

An acquisition thread gets values of base data items from a port for acquisition. For the periodic acquisition, a timer activates the thread, which reads the memory mapped device to get the base data item values. The acquisition period is determined according to the absolute validity interval of the base data items. For the aperiodic acquisition, the execution of the thread is initiated with an arrival of data to the queue implementing the port. To apply the RMA, an aperiodic acquisition task is regarded as a periodic task whose period is detemined by the minimum arrival interval fo the data. The acquisition thread calculates the values of derived data items from data item values which have been acquired. The acquisition thread put these data item values with a time stamp on the ring buffer, overwriting the oldest scene.

When a predefined number of scenes are acquired, the acquisition thread triggers the saving thread, which duplicates the scenes in a contiguous file before they are overwritten.

### 3.4.3 Method Execution Accoding to Requests

A request from a client is sent to the receiver thread with a socket based on the UDP/IP protocol. The receiver thread posts the request to the queue for the chief thread, which assigns the request to one of working threads.

When the periodic service is requested, a working thread is periodically activated to create a series containing a single scene which holds the latest acquired value. To enable for various periods to be specified, multiple working threads are prepared for the periodic service as a thread pool.

Since tasks for the on-demand service have no deadline, a single working thread is prepared for the service. When the chief thread posts a request to the queue for the on-demand service, the working thread starts to retrieve scenes so that it may create a series based on a specified provision schema in a specified rate.



### 3.4.4 Responses

There are requests of various priorities. To send their responses to clients according to the priorities, the prototype is equipped with more than one transmission queues, each of which has a unique priority. A series created by a working thread is sent to one of the transmission queues.

When several responses waits to be sent in a queue, they should be processed successively as far as there is no waiting response of a higher priority. If only a single transmitter was prepared for each transmission queue, a lower priority transmitter which is ready to sent a response could precede a higher priority transmitter which has several responses to be sent, when the latter transmitter finishes the transmission of one response. As a solution of this problem, a pair of transmitters are allocated to each transmission queue. When one of the pair finishes the transmission, the other one can immediatetely start the transmission. The pair for a transmission queue prevents a lower priority transmitter from preceding a higher priority transmitter.

### 3.4.5 Client Stubs

The interaction between RTDS and its clients based on the server-client model. To make use of the RTDS functions, clients call its stub routines. In the case of the periodic service, for example, a client first calls a stub routine to request RTDS of starting the periodic service. RTDS sends an acknowledement responce to the client if it can afford to serve. Next, the client waits for a series to be sent by RTDS, using another stub routine which will block its execution until a series arrives.

Since a socket based on the UDP/IP protocol is used for the communication between RTDS and a client, RTDS cannot send data of larger size than the limit determined by the protocol. When RTDS send a long series, it breaks the series into small chunks, from which a client stub routine assembles a series. Clients do not have to be conscious of the assembling, because client stub routines hide it.

Although RTDS can transmit series without any failure, clients in the information system layer do not necessarily get all the chunks. Note that clients in the information system layer may fail to finish tasks by their deadlines, because the clients work on non real-time operating systems such as UNIX. When a client fails to get a chunk, the chunk may be overwritten by a new one. Since RTDS cannot detect the receipt failure, it attaches a sequence number to each chunk sent to the client. The check of the sequence number in received chunks enables a client stub routine to find a failure in receipt of the preceding chunks. The client which fail to receive a whole series may issue a request to send the series again, if necessary.

## 3.5 Evaluation for Predictability

Here, an experiment result is presented to prove RTDS provides the predictability for the timing consistency. For the simplicity, let us refer to the number of data items in an acquisition schema as the acquisition size. The experiment investigates the schedulability of a task set in RTDS under various combinations of the acquisition size, the acquisition period, and the number of threads for the periodic service. First, the RMA is applied to predict the minimum acquisition period under which the real-time data server is schedulable for the given combinations of the acquisition size and the number of periodic service threads. The prediction is verified with an experiment on the prototype of RTDS.

### 3.5.1 Experimental Environment

A prototype of RTDS is constructed on a COTS real-time operating system, Lynx386, which is compliant with POSIX 1003.4. In this experiment, in order to make it easy to modify the acquisition size, an emulator thread is used to generate process values, instead of acquiring them from a plant. The emulator thread works in the same period as the acquisition thread. Each data item in the acquisition schema is 64bit in data length. Each periodic service thread provides a series which has a single scene consisting of three data items every second. The periodic service threads are created one by one to know when the prototype is not schedulable. The load of the periodic service threads is so small that the main memory



overflows before the prototype reaches its limit in the schedulability. Therefore, an extra looping load is add to each of the periodic service threads. The CPU time spent for one period of the periodic service is $11.2msec$ without the extra load and $161.2msec$ with the extra load. The working threads in this experiment are emulator thread $\tau_e$, acquisition thread $\tau_a$, saving thread $\tau_s$, receiver thread $\tau_r$, transmitter thread $\tau_{t0}$, $\tau_{t1}$, chief thread $\tau_c$, and periodic service thread $\tau_p$. We set the period of each periodic thread as

$$T(\tau_e) = T(\tau_a),$$

$$T(\tau_s) = 10 \cdot T(\tau_a),$$

$$T(\tau_p) = 1000msec,$$

where $T(\tau)$ is the period of thread $\tau$. Posting a series by $\tau_p$ to a transmission queue activates transmitter thread $\tau_{t1}$, which has to send all series within $T(\tau_p)$. Therefore, $T(\tau_{t1})$ is equal to $T(\tau_p)$. The minimum interval times[12] of $\tau_r$, $\tau_c$, $\tau_{t0}$ are regarded as $T_{msg}$ which is a minimum arrival interval of requests for starting the periodic service. In this experiment, $T_{msg}$ is set as $T(\tau_s) \leq T_{msg} \leq T(\tau_p)$.

Table 3.1 shows the CPU time, the period, and the priority of each thread when the acquisition size is 255. The CPU time means the time spent in the one period execution of threads other than $\tau_{t1}$. The CPU time for $\tau_{t1}$ means the time spent for the transmission of one series.

### 3.5.2 Prediction and Verification

In this experiment, the five kinds of threads are working; the acquisition thread, the saving thread, the periodic service threads, the transmitter threads to transmit series, and the emulator thread. Threads in a real-time data server are blocked only when they access the circular storage. The blocking time is small enough to be neglected.

Suppose that there are $m$ periodic service threads. A set of threads in the prototype is schedulable if

$$\frac{C(\tau_s)}{T(\tau_s)} + \frac{C(\tau_a)}{T(\tau_a)} + \frac{C(\tau_s)}{T(\tau_s)} + \frac{C(\tau_{t1})}{T(\tau_{t1})} + m \cdot \frac{C(\tau_p)}{T(\tau_p)} \leq (m+5)(2^{\frac{1}{m+5}} - 1). \qquad (3.1)$$

Table 3.1: CPU Time, Period, and Priority of Thread

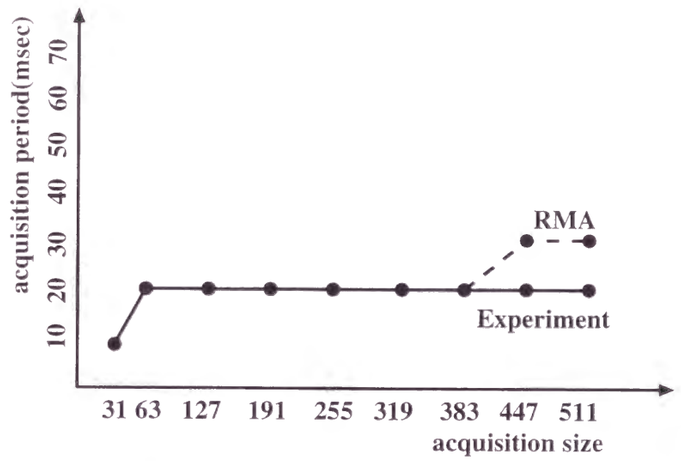
| Thread | | CPU Time (msec) | Period (msec) | Priority |
|---|---|---|---|---|
| Emurator | | 0.300 | 20 | 1 |
| Acquisition | | 0.840 | 20 | 2 |
| Saving | | 42.7 | 200 | 3 |
| Receiver | | 0.0150 | 1000 | 4 |
| Transmitter | Response | 0.0812 | 1000 | 6 |
| | Retrieval | 0.0812 | 1000 | 8 |
| Mother | | 0.0120 | 1000 | 5 |
| Periodic Reading | | 161.2 | 1000 | 7 |

The left side of the inequation corresponds to the utilization of the CPU[26]. When $m = 3$, the value of the right side of the inequation is 72.86%. Let $r$ the acquisition period. The values of the left side are 84.11% and 66.25% when $r = 10msec$ and $r = 20msec$, respectively. Therefore, the acquisition period which guarantees the schedulability of the real-time data server is $20msec$.

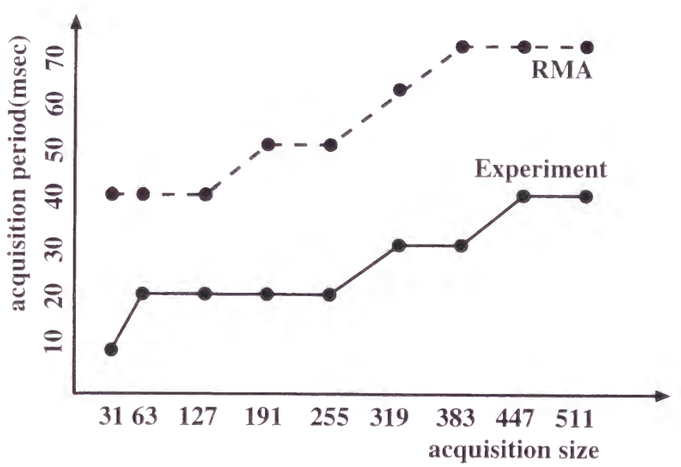
Similarly, the acquisition period under which the task set is guaranteed to be schedulable is calculated, letting the acquisition size be 31, 63, 127, 191, 255, 319, 383, 447, and 511. The calculation is done for two cases; in one case, the number of the periodic service threads is 3, and 4 in the other one. The dotted line in figure 3.4 shows the result of the calculation. The results presents the prediction of the acquisition period under which the prototype of RTDS will be schedulable. Since condition (3.1) represents a very pessimistic condition[46], an actual real-time data server is expected to be schedulable even in a shorter acquisition period than the predicted one.

The minimum acquisition period under which the task set in the prototype is actually schedulable is presented by solid lines in figure 3.4. The figure shows that the actual prototype of RTDS is schedulable in the acquisition period which is shorter than the predicted one. This verification reveals that RTDS can be designed with predictability, if the CPU time of each thread is known.






(a) In Case of 3 Threads for Periodic Service




(a) In Case of 4 Threads for Periodic Service

Figure 3.4: Prediction with RMA and Experimental Results

## 3.6 Server for Supervisory Control

This chapter explains the design of a prototype of Real-Time Data Server(RTDS). RTDS acquires data from plants without loss. When it receives request from a client, it works as a server to provide the client with the acquired data in the form of series.

The schedulability of RTDS can be predicted with the RMA, because the blocking time of threads is bound. The prediction of schedulability is verified with an experiment on an actual system.

The service of data according to requests during real-time acquisition is an essential function for DAS systems for supervisory control. Viewed from control systems, RTDS works as a data logger. It, however, works as a server from the view point of the clients in both real-time system layer and the information system layer.

In supervisory control, we have to note that reactions for state changes in a plant are indispensable functions, which are not included in RTDS explained in this chapter. In the next chapter, the prototype explained here will be improved to a middleware which provide active service as well as the passive service during the real-time acquisition.



# Chapter 4

# Real-Time Reactions According to Data Freshness

## 4.1 Reactions in Supervisory Control

Keeping real-time data acquisition abilities, many plant engineers want to enhance the DAS middleware systems so that they can detect predefined state changes in a plant for reactive works. There are many kinds of stage changes. Some of them require urgent reactions whose response time from the occurrence of the state changes to the notification must be short and bound. Others needs unpredictable time to detect and notify them. To detect any state change instantly, various conditions to detect specific changes should be evaluated every acquisition. The unpredictable consumption of the CPU power would degrade guaranteed acquisition ability, which is most important in DAS systems.

In industrial applications, it is important to detect state changes and react them. Varieties of timing consistency constraints in a data acquisition system are discussed in [32]. In [8], a static scheduling analysis is applied for real-time acquisition. Although these works deal with consistency in real-time tasks, they do not discuss the detection of specific state changes in a plant and reactions which follows the detection. The concepts of active databases[29][62] are useful for reactive tasks. However, as [40] and [54] point out, previous studies of active databases neglect real-time transactions. Although the sporadic server[56] can integrate the active tasks with real-time tasks, it is difficult to implement on COTS platforms. Since the

sporadic server assigns the available CPU time to each task according to its consumption. Its implementation needs modification of kernels.

Handling all tasks in a uniform manner would make it infeasible to give a guarantee for maintaining the timing consistency[40] [14] on COTS platforms. In this study, the timing consistency constraints are relaxed using the following characteristic in supervisory control; the fresher the data, the more strict the timing constraints in handling the data. Reflecting the characteristic specific to the domain, the essential mechanisms of the prototype explained in the previous chapter is enhanced to the *Active*RING model[52] which incorporates ECA mechanisms into circular storages. The *Active*RING model has the following features.

1. The *Active*RING model caches fresh data on the circular storages at any time.
2. The *Active*RING model provides an ECA mechanism cognizant of its time consumption.
3. The *Active*RING model sets deadlines according to data freshness.

The ECA mechanism in the *Active*RING model values ECA rules which uses only the latest data. The ECA mechanism executes these ECA rules after every acquisition. Some of them realize the urgent notification as firm real-time tasks, whose execution time can be bound. Others trigger reactions which need recent data. Soft deadlines can be assigned to the reactions, because the recent data are cached.

Based on the *Active*RING model, Real-Time Data Server(RTDS) is redesigned as a middleware to integrate reactions for state changes with real-time data acquisition. Many instances of RTDS work in numerous plants for steel mill, sewage disposal, and tunnel ventilation.

## 4.2 Characteristics in Real-Time Active DAS System

To integrate reactive works with real-time acquisition and service on COTS platforms, let us investigate characteristics of the tasks enumerated in table 2.1. The investigation for the acquisition and the passive service contributes to prototyping RTDS in the previous chapter.



Table 4.1: Characteristics of Tasks in DAS System

| Task | | | Handled Data | Value | Success |
|---|---|---|---|---|---|
| Acquisition | | | Latest scene | No value if it does not finish by its deadline | Necessary |
| Service | Passive | Periodic Service | Latest scene | No value if it does not finish by its deadline | Necessary |
| | | On-Demand Service | Series in an arbitrary duration | Value independent of the finishing time | Preferable |
| | Active | Real-Time Notification | Latest scene | No value if it does not finish by its deadline | Necessary |
| | | Best-Effort Notification | Recent series | Some values even if it does not finish by its deadline | Necessary |

Special attentions are paid for the active service to enhance RTDS to a real-time active DAS system.

### 4.2.1 Secure Acquisition

In DAS systems for plants, the most important task is acquisition of data item values. To maintain the absolute temporal consistency, acquired data item values must be so fresh that they correctly represent current plant status. In other words, it should be guaranteed that any acquisition task for a data item value is successfully completed by a deadline[see table 4.1]. Since the time stamp is attached to a scene at the time of the acquisition, it represents a transaction time. The difference of the transaction time from the valid time should be so small that they may be degenerated in practical usages of DAS systems. Deadlines for acquisition must be not only firm but also short.

### 4.2.2 Passive Service

According to requests from clients, the periodic service is activated every specified period. Suppose a controller which periodically determines its behavior from the latest input data.

A task in a DAS system must periodically transmit the latest acquired values of some continuous data items to the controller. The value of the task would be degraded to 0 unless every transmission finishes within its period. The periodic transmission must always succeed, so that the controller should work correctly. Since almost all clients request the latest scene for the periodic service, we restrict data used in the periodic service to the latest scene as shown in table 4.1.

The on-demand service activated by clients cannot handle reactive tasks. There are few urgent tasks for the on-demand service. Clients usually request to retrieve past series for the on-demand service, and rarely impose deadlines on retrieval of the series. For example, a program to improve product quality may analyze reference and feedback values for all billets milled during one month. Values of tasks in the on-demand service seldom vary on the completion timing of the tasks. Tasks to retrieve past data have no deadlines.

### 4.2.3 Real-Time and Best-Effort Notification

DAS systems are expected to detect state changes in a plant and notify clients of them with values of relevant data items. Since a failure of notification leads to wrong control activities, notification should always succeed. There are two kinds of reactions: urgent ones and ordinal ones. Through the development of numerous supervisory control systems, we have got the following empirical knowledge:

- real-time notification tasks can be assumed to use only the latest scene, and
- best-effort notification tasks would impart some values even if they do not finish by their deadlines.

The real-time notification is used for urgent reactions. Plants are generally equipped with specific sensors to detect significant state changes needing urgent reactions by means of only the current values of data items. Suppose a controlling system of a steel mill plant needs to be notified of billets' movement. When a billet moves, a specific sensor changes its output. The output is acquired as the latest value of a data item by a DAS system, which can detect



the state change. To react the state change immediately, current data item values relevant to it should be notified within a limited time.

There are also state changes which may be detected and reacted in a best-effort manner. Such changes are covered by best-effort notification tasks, which would usually handle not only the latest scene but also scenes around the change, i.e., a recent series. As an example of the reaction in a best-effort manner, suppose a controlling program which calculates the mill pressure with a mathematical model. Multiple recent value histories are accumulated in the model to prevent a deviated value history from affecting it. For following milling times, the program modifies the model with a history of feedback values related to the billet milled in this time. The completion of a milling activates a task to serve the feedback value history in the milling to the program. The service by the next milling imparts a great value, because the result of the nearest milling is reflected. The task, however, does not lose all of its value even if it does not finish until the next milling. An example of the detection in a best-effort manner is to find symptoms implying possible future abnormal states. The aim is to call operators' attentions. Through many experiences, operators know various symptoms, but it is an intolerable burden for them to keep staring a monitor not to miss any of the symptoms. As in an example in [48], some symptoms are represented by conditions specifying how a plant transits states as the time proceeds. Suppose a case where the temperature of a heated billet has not reached a target value, although the heater temperature has stayed in a preferable range for $m$ minute. DAS systems have to detect these symptoms using series in a recent duration.

The best-effort notification which finishes earlier imparts a greater value than ones which finishes later. The best-effort notification, however, should not be processed as a firm real-time task, because it would consume much CPU power to be used for the real-time notification. To avoid degradation of real-time notification ability, the best-effort notification should be processed in a different priority from that for the real-time notification.

### 4.2.4 Data Freshness

In plant supervisory control, fresh data are significant. The freshness depends on the difference of the acquisition time from the the current time. For the simplicity, any acquired data item is assumed to be included in a single acquisition schema. Let acquisition schema $s$ contain data item $d$. Suppose scene $\sigma_i$ which is associated with $s$ and whose time stamp is $time(\sigma_i)$. Let the current time be $C$. The value of $d$ recorded in $\sigma_i$ is fresher than the one recorded in $\sigma_j$ iff

$$\mid C - time(\sigma_i) \mid \; < \; \mid C - time(\sigma_j) \mid. \tag{4.1}$$

A DAS system acquires data item values from a plant and combines them with the time stamp to make a scene. It stores the scene as the latest one. On the other hand, applications in the information system layer requires the periodic service more frequently than the on-demand service. When the state of a plant changes, control systems needs temporally neighboring scenes around the change. Through many experiences, we have found that the access in a DAS system is strongly localized to fresh scenes.

Because of the locality of the reference, recent scenes should be cached to improve the access speed to them. As it is pointed in [32] and [45], a circular area which consists of two level storages is provided to store scenes according to their freshness. The latest scene which has been acquired overwrites the oldest scene on the circular area in the main memory. A bunch of scenes in the main memory is duplicated in one of contiguous files on a disk before they are overwritten. The two level circular storage allows to cache recent scenes and to store many past scenes for a long time. With a middleware which hides the conjunction of the two level circular storage, retrieval tasks can access scenes without being conscious of the physical location of a specific scene.

In a DAS system, it is also important to maintain the timing consistency. For this purpose, we apply the RMA to task scheduling in a DAS system. The task scheduling based on the RMA would be pessimistic without precise estimation of the execution time of each task. Firm deadlines should be assigned to tasks for the acquisition, the periodic service, and real-



time notification service. Since these tasks handles only the latest scene, we can precisely estimate the execution time. A DAS system, however, has to retrieve series which contain numerous scenes for the on-demand service and the best-effort notification service. It is not practical to assign a firm deadline to a task which retrieves an arbitrary number of scenes because the worst case execution time is tremendously large. Neither the on-demand service nor the best-effort notification service should not be realized with a firm real-time task. These tasks should be processed in background. This scheduling scheme causes any problem for tasks for the on-demand service, because they seldom impose deadlines. For the best-effort notification service, soft deadlines should be assigned to tasks. Recent scenes which are necessary in the best-effort notification service are cached in any time. The RMA which is pessimistic in the scheduling leaves much CPU power for background processing. The execution time of tasks for the best-effort notification service is so short that they are rarely preempted by tasks of higher priorities. The response time in the best-effort notification service can be estimated with the number of handled scenes in advance. We can statistically guarantee that a best-effort notification task finishes by a soft deadline.

Making best use of the characteristics mentioned in this section, we construct a DAS middleware in the following way.

- Recent scenes are cached.
- Deadlines are assigned to tasks according to the freshness of scenes handled by the tasks:
  - firm deadlines for tasks to handle the latest scene,
  - soft deadlines for tasks to retrieve recent series, and
  - no deadlines for tasks to request past series.

## 4.3 Model Based on Data Freshness

Reflecting the characteristic specific to DAS systems, the active service is integrated with the real-time acquisition and the passive service. The essential mechanisms of the prototype

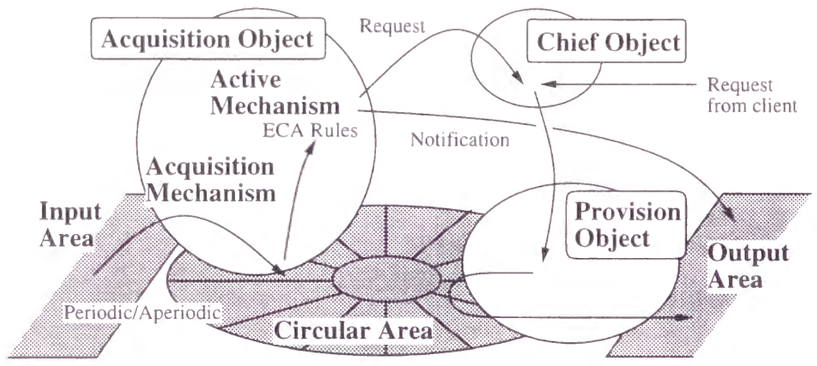



Figure 4.1: *Active*RING Model

of RTDS explained in the previous chapter is enhanced to a computational model for a real-time active DAS middleware.

### 4.3.1 *Active*RING Model

As a computational model for a DAS middleware, we propose the *Active*RING model, which cashes recent scenes and assigns a deadline to a task according to the freshness of scenes handled by the task. The *Active*RING model incorporates an active mechanism into a circular storage.

As figure 4.1 depicts, the *Active*RING model consists of areas which hold temporal data and objects which has methods to realize the functions shown in figure 2.1.

There are three kinds of areas: input areas from which data are acquired, circular areas in which scenes are stored, and output areas to which series or data calculated from them are written. We assume that data from the real-time system layer are written in the input areas, maintaining the absolute temporal consistency. For example, the input area can be implemented either by a memory-mapped device to hold data sent from sensors like an interface board of the data highway[44] or by a queue to receive messages from a sensor. Values acquired from the input area constitute a scene combined with a time stamp which represents the acquisition time. A circular area holds continuous scene arranged according



to their time stamps, with the oldest scene overwritten by the latest one. In plants, values of various data items are acquired. A single circular area is prepared for continuous data items which have the same acquisition period. Circular areas for discrete data items are separated from ones for continuous data items, because values of discrete data items are acquired aperiodically. Output data for clients are written on the output areas.

The *Active*RING model has three kinds of objects. It may have multiple acquisition objects and provision objects. There is a single chief object which assigns a retrieval request of a series to one of provision objects. Once the chief object assigns a request to a provision object, it interferes with neither acquisition objects nor provision objects. Acquisition objects and provision objects repeat a sequence of reading data from one area, computing new data if necessary, and writing data to another area. The flow of data is one way; read operations are never interleaved with any write operation. Note that the two kinds of objects share only data in circular areas. As long as they make no access to circular areas, they can work independently.

In the *Active*RING model, a circular area is associated with an acquisition object. In each acquisition object, an event activates a method to acquire data item values based on an acquisition schema. Events for periodic acquisition are sent from a timer, while arrival of data from outside to a queue is regarded as an event to activate aperiodic acquisition. An acquisition object has an acquisition mechanism and an active mechanism. The former is realized with acquisition method tailored for a specific acquisition schema, which consists of base data items and derived data items. The acquisition mechanism acquires base data item values from the input area and calculates derived data item values from them. Like a differentiation of the present value from the previous value, the acquisition mechanism may derive current data item values with scene which have been acquired. The active mechanism based on the ECA model[29][62] reacts state changes in a plant. An active mechanism contains ECA rules, each of which is a pair of a condition on the latest acquired data and an action to be executed when the condition holds. The conditions of all ECA rules are evaluated every acquisition, to find state changes immediately. When a condition holds, an action associated with it is executed to notify the state change of clients. The state change

is notified in either of the following ways. The active mechanism writes data calculated with the latest scene into an output area for itself, or it issues a request to the chief object. The former directly notifies the client of the state change, while the latter causes one of provision objects to retrieve a series around the state change so that the client may be notified of the state change along with data calculated with the series.

On reception of a request, the chief object chooses a provision object and forward the request to the provision object. If it receives a request for periodic service of the latest scene, the chief object sets a timer to activate a provision object every specified period.

A provision object has methods specific to provision schemata. On receiving a request, a provision object executes one of methods to retrieve a series based on a provision schema specified in the request. An output datum calculated with them is written into an output area. More than one provision objects are prepared so that requests with different priorities can be processed simultaneously.

### 4.3.2 Time Cognizant Notification

Notification in the *Active*RING model is cognizant of time consumption in evaluation of conditions and execution of actions. Some notification tasks use only the latest values of data items, while others need value histories of data items related to specific change. ECA rules for the latter would need series. The evaluation time cannot be bound for a condition on series in an arbitrary duration. It is impossible to estimate the execution time of an action which needs series. If the active mechanism execute all of the ECA rules every acquisition, the consumption of the CPU power would degrade guaranteed acquisition ability, which is most important in DAS systems.

To solve this problem, we distinguish ECA rules which uses only the latest values from others. For such rules, we can precisely estimate the time which is necessary to evaluate the conditions and execute the actions. We refer to the conditions of these rules as *on-acquisition-conditions*. Only when on-acquisition-conditions hold, the *Active*RING model executes notification tasks which need series. With on-acquisition-conditions, the *Active*RING model



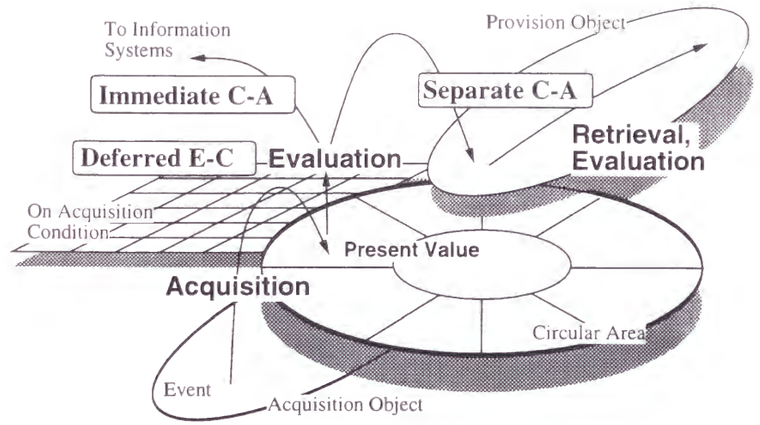



Figure 4.2: ECA Coupling Modes in Active Mechanism

finds a change which should be notified to clients in a real-time manner, as well as it triggers best-effort notification. The active mechanism evaluates all the on-acquisition-conditions every acquisition. We assume an on-acquisition-conditions can be represented by a linear (in)equality on the newest values of data items in a single circular area. The assumption is quite natural because a plant should be equipped with a specific sensor for real-time notification. The evaluation time of each on-acquisition-condition can be bound with the number of data items used in the (in)equality.

The active mechanism in the *Active*RING model is shown in figure 4.2. A timer sends an event to activate the periodic acquisition. Arrival of data to the queue is an event to activate the aperiodic acquisition. The evaluation of on-acquisition-conditions is delayed until all data item values have been acquired, because it may use arbitrary data item values. If any of the on-acquisition-conditions holds, the action is either

- to write data into output area for a specific client, or
- to issue a request to a provision object through the chief object.

The former corresponds to the real-time notification, where the active mechanism notifies a

client that the condition holds for itself. Data used in the notification are calculated with the latest values of data items. The real-time notification is realized with the E-C coupling of the deferred mode and the C-A coupling of the immediate mode[29]. The best-effort notification which needs series is achieved by the cooperation of an acquisition object and a provision object. When an on-acquisition-condition holds, the active mechanism can issue a request to retrieve a series. Through the chief object, the request is sent to the provision object. The provision object performs retrieval of series succeeded by condition evaluation and action execution using retrieved series. The best effort notification is realized with the E-C coupling of the deferred mode and the C-A coupling of the separate mode. Since the retrieved series are related with the change which has just happened, they are expected to be recent ones, which are cached on a circular area in the main memory. They can be retrieved quickly.

### 4.3.3 Maintaining Consistency

Since scenes are retrieved during acquisition, an appropriate exclusion method is necessary to arbitrate simultaneous access to a circular area. As it is explained in the previous chapter, the *Active*RING model locks only the management tuple indicating what position is being overwritten[50], with a semaphore based on the priority inheritance protocol[39]. The exclusion method leads to a small and bound blocking time. We can apply the deadline monotonic analysis[26], which is a simple extension of the RMA, to schedule tasks having firm deadlines. Scheduling policies based on the deadline monotonic analysis are easily implemented on COTS real-time operating systems.

The *Active*RING model can process tasks shown in figure 2.1. Using circular areas and the deadline monotonic analysis, we can guarantee all firm real-time tasks always finish by their deadline. The deadline for each acquisition task is determined according to the absolute validity interval of the base data item so that the absolute temporal consistency may be maintained. The absolute validity interval also determines the deadline for each real-time notification task. The calculation of derived data item values uses data item values which



Table 4.2: Deadlines According to Data Freshness

| Task | | | Deadline | Guarantee |
|---|---|---|---|---|
| Acquisition Task | | | Firm | Static |
| Service | Passive | Periodic Service Task | Firm | Static |
| | | On-Demand Service Task | No Deadline | —— |
| | Active | Real-Time Notification Task | Firm | Static |
| | | Best-Effort Notification Task | Soft | Statistical |

have been acquired within the relative validity interval of the relevant data items to maintain the relative temporal consistency. Details have been explained in section 2.4.3.

### 4.3.4 Deadlines According to Data Freshness

The deadline monotonic analysis[26] is applied for the schedulability check of a task set in the *Active*RING model. All tasks are treated as periodic tasks, each of which has a static priority. As for a task which aperiodically arrives and must finish within its deadline, the priority is assigned based on its minimum arrival interval. An aperiodic task with no firm deadline is processed in background.

The strictness of a task deadline is determined according to the freshness of data handled in the task[see table 4.2]. Deadlines in the acquisition are firm ones, because it always handles the latest values. In the passive service, firm deadlines are set to periodic service tasks to provide the latest values, while on-demand service tasks have no deadline.

In the active service. the separation of rules having on-acquisition-conditions from others contributes to setting deadlines according to the data freshness. Since the execution time of such rules are bound, the real-time notification which uses only the latest values can be realized by a task having a firm deadline. On the contrary, the best-effort notification

which may need recent value histories is realized with a provision object triggered by the active mechanism. It would be hard to predict the response time for tasks in the provision mechanism, because they are processed in background. However, the deadline monotonic analysis is so pessimistic that much CPU time remains to process background tasks. Unless background tasks make access to disks, the processing time of them is not so long that they are preempted many times. Meanwhile, most of the tasks for the best-effort notification make access to scenes acquired around the change which has just occurred. Since such scenes are expected to be cached, the response time does not differ so much from an estimation from the number of retrieved scenes. We can give a statistical guarantee to the tasks triggered by the active mechanism to be processed in the provision object. The deadlines of best-effort notification tasks are regarded as soft ones.

## 4.4 Middleware

Based on the *Active*RING model, the prototype of Real-Time Data Server(RTDS) explained chapter 3 has been impoved as a DAS middleware. It acquires temporal data from plants and, at the same time, provides clients with both active and passive services. Here, the design is explained. In the explanation, outputs of RTDS are limited to series consisting of more than 0 scenes, which are transmitted to clients with sockets based on UDP/IP protocol.

### 4.4.1 Components

Figure 4.3 depicts an implementation of RTDS which consists of areas and objects. Circular areas stores recent scenes on ring buffers in the main memory, and past ones in contiguous disk areas. An object may have multiple threads. Each thread repeatedly executes a method either periodically or aperiodically. The objects work as follows.

- Acquisition objects — Each of them has an acquisition thread to acquire scenes and a saving thread to duplicate the scenes in a contiguous file before they are overwritten. After the acquisition, the acquisition thread evaluates on-acquisition-conditions.



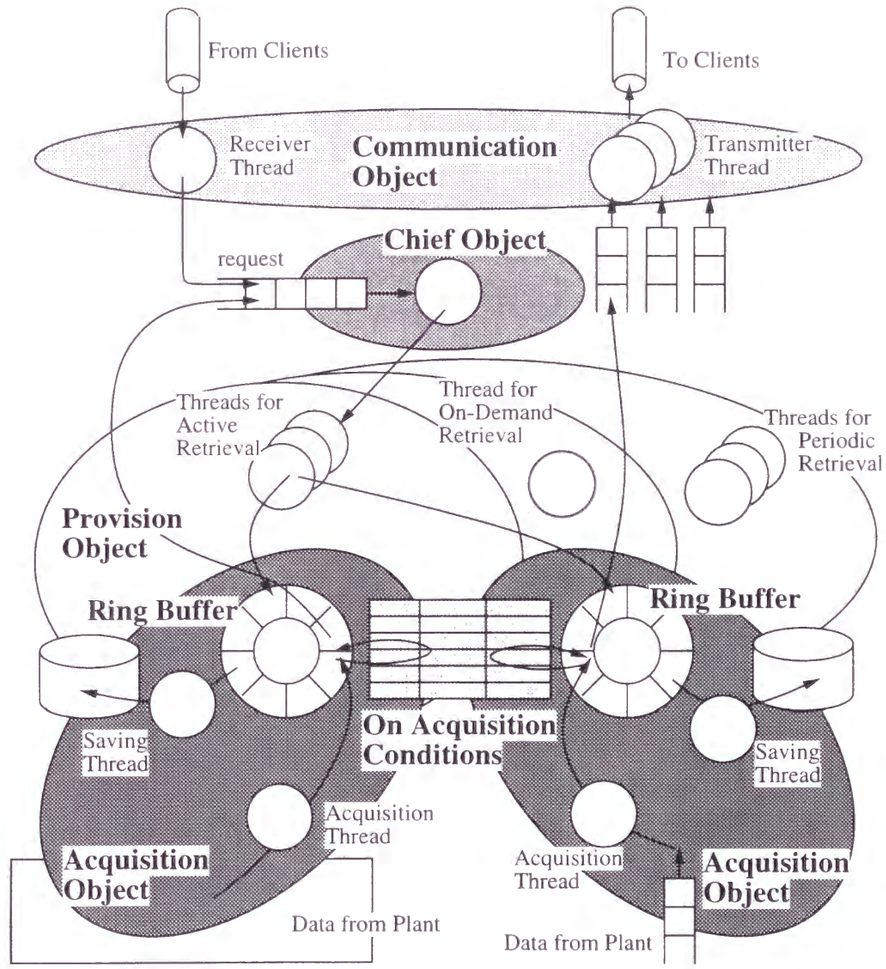



Figure 4.3: Component of RTDS

- Provision objects — A provision object may have a thread pool. When a request arrives. one of threads is activated for the request. Some threads periodically read the latest scene from ring buffers, while others retrieve scenes in specified durations. As figure 4.3 shows, the provision object for the periodic service or the best-effort notification has a thread pool to process multiple requests simultaneously. The provision object for the on-demand service has a single thread. Tasks for the on-demand service which have no deadline is processed one by one, to reduce conflict of disk access for the on-demand service with the saving of acquired scenes.

- Chief object — Any request from clients or the active mechanism is sent to the chief object, which assigns each request to an appropriate provision object.

- Communication object — Data alignment and byte order varies with machines and operating systems. To hide the differences, RTDS is equipped with a communication object. It communicates with clients by data in a format independent of machines and operating systems. A receiver thread in the communication object converts requests from clients into data native to RTDS, while a transmitter thread sends series to clients in the independent format.

### 4.4.2 Middleware between OS and Applications

The implementation of RTDS requires the following operating system services:

- a fix priority based task scheduling service which makes all tasks fully preemptable,
- a semaphore service based on the priority inheritance protocol, and
- a timer service.

Since these services are supported in many COTS real-time operating systems, RTDS can be implemented on multiple platforms.

If the services are supported, RTDS can provide the functions shown in tablle 2.1 with its clients. Figure 4.4 illustrates the relationships among the functions, the implementation



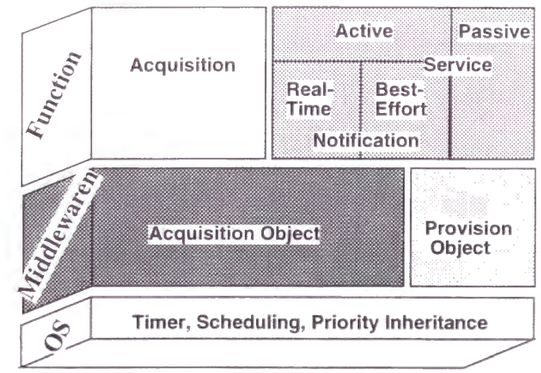



Figure 4.4: OS, Middleware, and Applications

of RTDS, and the OS services. The Acquisition, which is the most important function in RTDS, is realized by acquisition objects. The passive service is achieved by provision objects. The realization of the active service is founded on acquisition objects and provision objects. The real-time notification is realized solely by acquisition objects, while the best-effort notification is accomplished with the cooperation of acquisition objects and provision objects.

### 4.4.3 Interfaces of Middleware

RTDS has simple interfaces so that it may be used as a middleware in a various kinds of plants. As interfaces to acquire data, RTDS provides shared memories and queues. For clients, RTDS prepares client stub routines.

When RTDS uses a shared memory as an input interface, the shared memory is divided into a lot of small sections, each of which corresponds to a base data item. RTDS converts a value on a section into a data item value with a predefined function. A queue is used to acquire data which occur occasionally. Data of various structure can be enqueued, because there are many kinds of occasional data. RTDS assumes that the top field of enqueued data holds a type identifier to indicate its structure. With the type identifier, RTDS interprets the contents of enqueued data.

To make use of the RTDS functions, clients call its stub routines. In the case of notification, for example, a client first calls a stub routine to inform RTDS that it is ready to be notified. Next, the client waits to be notified by RTDS with another stub routine which will block its execution until the notification.

### 4.4.4 Condition and Action in Notification

A circular area has a table for ECA rules. Each entry of the table has a condition field and a message field.

The condition fields specify on-acquisition-conditions, which are evaluated by the acquisition thread after every acquisition. Each data item is not referred to by name, but by an offset from the top address of the scene. Let $b$ and $d_i$ denote the top address of a scene and the offset of $i$-th data item, respectively. Any on-acquisition-condition in the condition field is represented by a linear (in)equality on the latest values of data items as follows.

$$a_{min} \leq \sum_{i=1}^{n} a_i \cdot content\_of(b + d_i) \leq a_{max}, \tag{4.2}$$

where $n, a_{min}, a_i, a_{max}$ are the number of data items, the lower bound, the coefficient of the $i$-th data item, the upper bound, respectively. The maximum evaluation time for the inequality can be bound with $n$.

For example, assume that the latest scene in a circular buffer has data item $wc$ whose value is 1 if a billet is going through a mill, and 0 otherwise. In addition to that, let us assume that the same scene has $wp$ which holds a copy of the value of $wc$ in the previous acquisition time. The exit of the billet can be represented by $1 \cdot content\_of(b+d_{wc}) - 1 \cdot content\_of(b+d_{wp}) < 0$.

The message fields specify the actions performed by the active mechanism when the on-acquisition-conditions hold. For the real-time notification, the active mechanism posts a series consisting of the latest scene to a queue of the communication object. The other action is to send a request to the chief object so that the provision object may work for best-effort notification.



## 4.5 Evaluation for Reactions

Here, we demonstrate that RTDS reflects deadlines according to data freshness. In this paper, we consider two kinds of deadlines in RTDS: firm deadlines and soft deadlines. In sectin 3.5, it is shown that firm real-time tasks always finish by their deadlines if the task set in RTDS is judged to be schedulable with the deadline monotonic analysis. Soft deadlines are set for best-effort notification tasks triggered by the active mechanism to be processed in the provision mechanism. To verify that these tasks finish within their soft deadlines, a prototype on a computer which has a 133MHz Pentium processor and a PCI SCSI disk is constructed using LynxOS, a COTS real-time operating system. The prototype is a simplified steel mill plant system, which has two acquisition objects: an aperiodic one and a periodic one which acquires data every 100 msec. Each of the acquisition objects acquires a scene whose size is 512byte, and saves 2 scenes every 2 acquisition. The ring buffer of each acquisition object caches 256 scenes.

As the most typical best-effort notification task, we adopt tasks to retrieve scenes around a detected state change. The tasks retrieve scenes acquired periodically. The active mechanism issues a request of the retrieval just after it finds a state change. Suppose a state change occurs at time point $t$ and $n$ scenes are retrieved. Let $t_e$ and $t_c$ be the time point on which the active mechanism issues a request and the time point on which the client gets a series consisting of retrieved scenes, respectively. In the experiment, the retrieval time, which is the time difference from $t_c$ to $t_e$, is measured 2500 times for requests to retrieve all scenes in $[t-(n-1)\cdot r,\ t]$, where $r$ is 100msec. We measure the retrieval time, setting $n$ to a random value among [1, 3000].

Each of 2500 trials is plotted in figure 4.5. If $n$ is greater than 256, a task reads scenes not only from the ring buffer but also from the disk. As the graph shows, the deviation of the retrieval time is very large if a task retrieves scenes from the disk. The phenomenon can be explained as follows. It takes a long time to process a retrieval task from the disk. During the processing time of the retrieval task, there can be several opportunities when acquired scenes are duplicated in the disk. Any retrieval task yields to tasks of higher priorities, if the

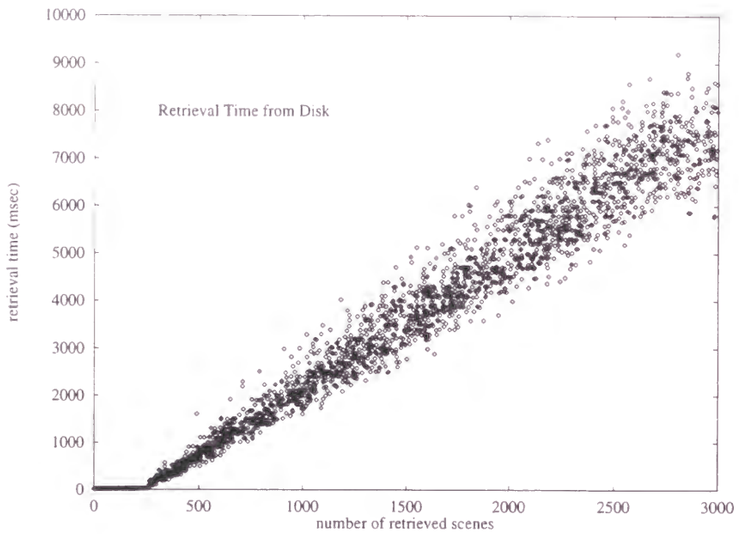



Figure 4.5: Retrieval Time from Disk

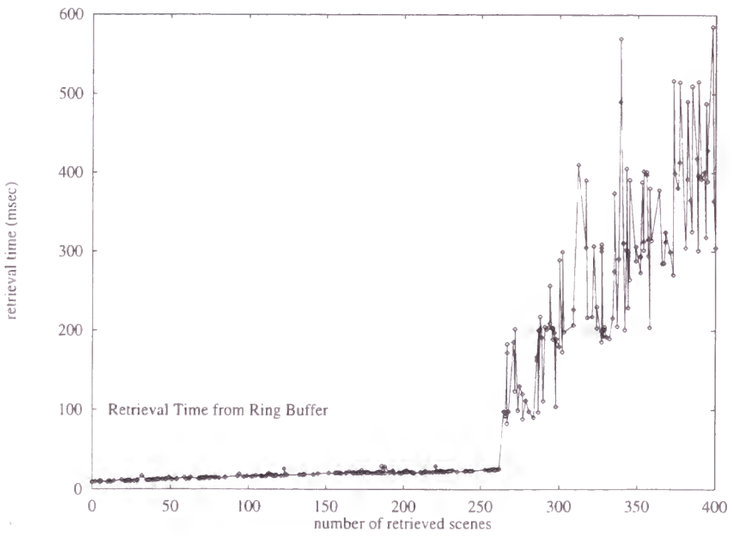



Figure 4.6: Retrieval Time from Ring Buffer



latter wait for disk access. Since tasks to duplicate scenes in the disks have higher priorities, the retrieval is preempted many times, which causes the deviation.

On the contrary, the deviation of the retrieval time is very small if a task retrieves scenes only from the ring buffer. An enlarged graph of the previous one is figure 4.6, where the number of retrieved scenes is less than 400. As the graph shows, the approximate retrieval time of scenes from the ring buffer can be estimated using the number of retrieved scenes. The deadline monotonic analysis is so pessimistic that much CPU time remains for processing of background tasks. Since a retrieval task from the ring buffer is completed in a small time, other tasks seldom preempt the task. It is scenes relevant to detected state changes that used in notification. Since the state changes have just occurred, all these scenes are expected to stay on the ring buffer. Therefore, the retrieval time of scenes can be estimated in the best-effort notification service. The experiment statistically proves that soft deadlines are set for the best-effort notification service, even though they are processed in background.

## 4.6 Integrating Active Service with DAS Middleware

The *Active*RING model is proposed to integrate the active service with a DAS system.

The *Active*RING model reflects the characteristic in the supervisory control. Since fresh data are frequently accessed, recent scenes are cached in circular areas. To react state change in a plant, each circular area is equipped with the active mechanism, which separates ECA rules whose execution time is bound from others. The *Active*RING model sets deadlines according to data freshness; firm real-time tasks treat the latest scene, soft real-time tasks process recent series, and non real-time tasks retrieve past series. The *Active*RING model enables the integration with COTS technologies. Based on the *Active*RING model, Real-Time Data Server(RTDS) is evolved to a DAS middleware. An experimental result is explained to show that RTDS provides service according to data freshness.

Instances of RTDS now work in sewage plants and steel mill plants. Others have been applied to ventilation systems for tunnels.

# Chapter 5

# Plant Operations Based on Trend Recognition

## 5.1 Varieties in Operation Responses

To show an important application of time dependent data, an expert system using series[48] is explained in this chapter. A lot of effort has been made to put expert systems for plants into practice like a work in [33]. A practical distributed AI system[18] explained in this chapter aims to guide plant operations in a steel-galvanizing plant where a steel plate is coated electrically. In the steel-galvanizing plant, coating weight is regulated with several decades of plant variables. The modification of one plant variable brings its effect in several seconds, while the modification of another in several hours. The former is referred to as a fast plant variable, and the latter as a slow plant variable. Fast plant variables are often modified in an emergency, because its immediacy is effective. The modification of fast plant variables, however, often prevents the preceding modification of a slow plant variable from producing an expected effect. In the steel galvanizing plant, a model to take all plant variables into consideration is hard to be established, because of difficulties in predicting an effect of operations on slow plant variables. Although most of plants are well modeled for regulation, proper mathematical model has not been established for the steel galvanizing plant.

In spite of lacking a proper mathematical model, experienced operators can regulate the

coating weight manually. They distinguish the plant variables by whether they bring their effects immediately or not. The experienced operators use a fast plant variable for the regulation of the coating weight, while a slow plant variable for compensation of fast ones. From their experiences, they can select the appropriate moment at which fast plant variables would be compensated.

Using AI technologies, a plant operation supporting system is developed to give operation advice competing with the experienced operators. In the system, the tasks of the operators are accomplished by multi agents[12]. An agent applies rough mathematical models to fast plant variables to regulate the coating weight. Other agents suggest to modify slow plant variables to compensate the fast plant variables. To select the appropriate moment to modify the plant variables properly, the compensation agents have to monitor entities in the plant, being conscious of the course of time.

This chapter uses series as a framework which can represent the course of time. Each agent investigates series in an appropriate form for a task assigned to it; it uses the reduced series which abstract states of a monitored entity. Each agent recognizes not only a current state but also a state transition in the course of time to select the best moment for a plant operation. The recognition of the state transition needs the length of intervals during which the entity stays in specific states, which is not addressed in conventional studies on temporal relationships[1][30][53]. An experiment in a real plant has proved that the developed plant operation supporting system can guide plant operations in much the same way as an experienced operator.

## 5.2 Expertise in Steel Galvanizing Plant

### 5.2.1 Why AI System?

In the steel galvanizing plant, coating weight is regulated with several decades of plant variables. Some of them are shown in table 5.1. For each plant variable, the table indicates both a time-lag of effect appearance in the feedback value from modification of the reference



Table 5.1: Response Time of Plant Variable

| plant variable | difference in modification | feedback from modification | | coating weight from feedback |
|---|---|---|---|---|
| | | increment | decrement | |
| current density | any value | several msec | several msec | several sec |
| line speed | any value | several sec | several sec | several sec |
| pH | 0.1 | several min | several hour | several min |
| temperature | $1°C$ | several min | several decades of min | several min |
| concentration | 0.1 $mol/l$ | several hour | several hour | several min |

value and a time-lag of effect appearance in the coating weight from the effect appearance in the feedback value. We refer to the time of the effect appearance in the coating weight from the modification of the reference value of a plant variable as a response time of the plant variable. As the table illustrates, the response time varies from several seconds to several hours. A plant variable whose response time is short is a fast plant variable, while a plant variable whose response time is long is a slow plant variable.

In plant operations using the plant variables of various response times, it is very difficult to modify a slow plant variable. Operators have to wait for a long time until modification of a slow plant variable brings an effect. A fast plant variable is usually modified many times during the waiting time. Thus, the modification of the slow plant variable does not always bring an effect as it was expected. The steel galvanizing plant is usually operated manually without a proper mathematical model which takes all plant variables into consideration. The plant cannot be operated without experienced operators who can properly use several rough mathematical models and rules acquired from their experiences according to specific conditions.

The experienced operators have acquired expertise on which they properly use plant variables of various response times. In this study, the expertise is represented as heuristic rules.

### 5.2.2 Coating Control and Its Compensation

Many interviews with experienced operators have proved that they operate the plant on the following way.

1. They classify plant variables into three groups by their response time. The first one is a group of short response times (several seconds). The second one is a group of intermediate response times (several minutes), and the last one is a group of long response time (several hours). We refer to plant variables of each group as fast plant variables, moderate plant variables, and slow plant variables, respectively.

2. They regulate the coating weight with the fast plant variables.

3. They compensate the fast plant variables with the moderate plant variables, and the moderate plant variables with the slow plant variables.

The third item is necessary, because every plant variable can take its value only in a specific range. For example, operators try to raise the present value of a fast plant variable, when it is effective in keeping the coating weight in a desirable range. It, however, cannot be raised if it is close to the upper limit. An experienced operator would have decreased the value in advance. The experienced operator would have modified a plant variable which has a longer response time, expecting that the modification would lead the plant to a situation where the value close to the upper limit can be decreased. He would have selected the appropriate moment for the compensation, examining whether an expected effect of the modification appears in a specified time.

### 5.2.3 Agents Reflecting Expertise

Figure 5.1 depicts the developed plant operation supporting system in which three agents act concurrently. `CoatingControl` is an agent which searches a plant operation to regulate the coating weight with a short plant variable such as the current density and the line speed. `PrimaryCompensation` is an agent to compensate the fast plant variables with moderate plant variables such as the temperature and pH. `SecondaryCompensation` is an agent to compensate the moderate plant variables with slow plant variables such as the concentration of the solution for galvanization.

In figure 5.1, plant variables which enter into an agent from its left side and exit to its right



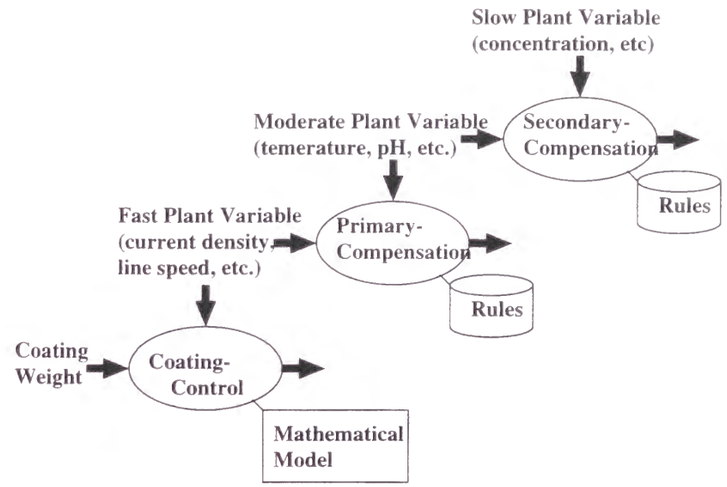



Figure 5.1: Agents for Regulation and Compensation

side are controlled variables regulated with plant operations suggested by the agent. Plant variables which enter into an agent from its upper side are manipulated variables whose modification the agent suggest. Operators modify the reference value of the manipulated variable, expecting the modification would lead the feedback value of a controlled variable to a desirable range.

## 5.3 Framework Conscious of Time

Experienced operators monitor not only the present state of an entity in the plant, but also state transitions to the present time. A series is used to represent a state transition of a monitored entity in the course of time. A framework is necessary to represent characteristics in it.

### 5.3.1 Instance and Schema

A set of data item values acquired from a monitored entity defines a state of the entity. By specifying the data items, a schema defines a monitored entity in the plant operation sup-

```
schema tank {
    int coat;          /* feedback value of coating weight */
    int LS;            /* feedback value of line speed */
    int LS_upLim;      /* upper limit of line speed */
    int LS_lowLim;     /* lower limit of line speed */
    int tmp;           /* feedback value of temperature */
    int tmp_chg;       /* difference of reference value of temperature */
};
```

Figure 5.2: Schema for Tank

porting system. An instance associated with schema `tank` shown in figure 5.2 can represent a state of a monitored tank at every acquisition time. A state of the tank is represented by the feedback value of the coating weight, the line speed, the upper limit of the line speed, the lower limit of the line speed, the feedback value of the temperature, and the difference of the reference value of the temperature from the previous one.

In this example, the tank has the line speed limits which varies with the time. To represent the limits, extra data items, `LS_upLim` and `LS_lowLim`, are declared in the schema. The relationships between these limits and the line speed are represented by conditions explained the succeeding section.

### 5.3.2 Skeletons

Let us refer to a characteristic of state transitions as a trend. Experienced operators make their decisions based on trends. They distinguish series which match specific conditions from others.

Two kinds of conditions are provided to represent trends. A restriction is the range to which the value of a data item should belong. A constraint is a linear inequality more than one characteristic variables should satisfy. An example is shown in figure 5.3, where restrictions are conditions following "`such that`" and a constraint is a condition following "`under`". The constraint represents the relationship between the line speed and its limits.



```
concept skeleton normal tank {
 means
  tank
    such that coat: >= 130, <= 150;
}

concept skeleton reducableLineSpeed tank {
 means
  normal tank
    such that coat : >= 145;
    under LS_upLim - LS < 20;
}

concept skeleton tmpInc tank {
 means
  tank
    such that tmp_chg >= 0;
}


series skeleton toBeRepaired tank {
 in 45 seconds;
 means tank is
  at(0)  tmpInc;
  from(0) to(30) normal;
  from(30) to(45) reducableLineSpeed;
}
```

Figure 5.3: Skeleton for Tank

Using restrictions and constraints, a concept skeleton defines a condition on instances of a specific schema. It corresponds to all instances which satisfy the conditions. Concept skeleton "`reducableLineSpeed`" in figure 5.3 defines the condition required for the line speed to be reduced.

A series skeleton is a condition on series of a specific concept. It is a sequence of concept skeletons accompanied with the length of intervals where the instances should satisfy them. It has a time-axis. A starting terminal point and an ending terminal point of each interval are specified with the distance from the basis of the time-axis. A sequence of periods in a series skeleton represents the course of time. Series skeleton "`toBeRepaired`" in figure 5.3 indicates state transitions of the tank after increment of the reference value of the temperature. It specifies that the line speed should be reduced because the preceding plant operation brings an expected effect; after the increment of the temperature, the tank stays in states associated with "`normal`" for about 30 seconds, and in states associated with "`reducableLineSpeed`" for about 15 seconds. An appropriate moment to properly modify the line speed is selected using this series skeleton. A sequence of periods is convenient to recognize a time required in a chemical reaction.

### 5.3.3 Matching

A series skeleton represents a trend or an effect of a preceding plant operation. The unification of a series with a series skeleton allows us to recognize a series. The unification consists of two kinds of matching.

Scene matching is to check which concept skeletons specified in a series skeleton are satisfied by an instance state recorded in each scene. The scene matching clarifies a period where each concept skeleton are satisfied. The time-axis matching is to check whether the clarified period meets the periods specified in the series skeleton.



## 5.4 Agents using Series: Specialist Modules

### 5.4.1 Facilities of Specialist Modules

Specialist modules are introduced as agents to support plant operations. General features of agents based on distributed AI methods are explained in [18]. The features include the followings.

**Concurrency** A target problem is solved with concurrent execution of many tasks. Several agents act simultaneously for the execution. Each agent is autonomous.

**Modularity** Activities of agents are encapsulated in object-oriented manners. Agents have procedures and local memories hidden from outside. They communicate with each other using messages[16]. The interaction between them are limited to the message passing.

**Specialization** They have heuristic rules to carry out the tasks assigned to them. They would accomplish only tasks relevant to them, and neglect others. The target problem is solved with cooperation of them.

Specialist modules are based on agents that are proposed in MACE[12], but they have functions specific to the recognition of series. Each specialist module can keep series of a specific schema in its local memory. When it receives a message whose content is a newly acquired scene, it couples the scene to the back of the internal series. If the series too long to be held in the specialist module, an early part of the series is stored in the external file, and an entry of the file is registered in the specialist module. The entry is a key to retrieve the early part of the series. Each specialist module contains procedures called as methods which are activated by either external messages or events such as timer alarms. It can set a timer to activate a method for a regular recognition of its internal series. In this method, it compares the series with several series skeletons to examine whether the series matched any trend. It selects a plant operation according to results of the matching.

The whole task for supporting plant operations is distributed to several specialist modules. An operation proposed by a specialist module should not conflict with operations proposed by others. Specialist modules pass messages to solve conflicts as described in section 5.4.3.

### 5.4.2 View of Specialist

A plant operation supporting system should consist of efficient components so that it may work as a real-time system. Each specialist module should record the state transition of the monitored entity in an efficient way to carry out a task assigned to it.

For example, experienced operators often make their decisions by examining only whether the temperature is increased or decreased by more than $1°C$. What should be recorded is one of "`increment`", "`decrement`", and "`no operation`". Experienced operators would examine the entity state not at the figure level but at the category level. Categories of states can be represented by concept skeletons.

This study values that a series is recognized by two stages. The scene matching stage determines concept skeletons which an instance matches with. The record of indices of matched concept skeletons would make time-axis matching possible. Let us consider a new schema which declares data items representing the matched concept skeletons. This schema is acquired when the original concept is viewed from an angle relevant to the task assigned to the specialist module. The schema is referred to as an aspect of the original schema.

An aspect strongly depends on a task. Each specialist module increases its efficiency, viewing the state transition of the monitored entity in a way specialized to its task. An aspect is the reduction of an original schema to a part relevant to the task. In the actual observation, the instance state seldom transit categories defined by concept skeletons. The aspect would greatly compress the length of series.



### 5.4.3 Coordination with Future Series

Several specialist modules independently selects a plant operation. A plant operation of one specialist module may conflict with that of another specialist module. A conflict means more than one plant operations on one plant variable in a specific future period. An arbitration mechanism is needed so that each specialist module may not conflict with others in plant operations. Conflicts must be solved in all future periods. Dechter proposed a temporal constraint network[7] to maintain the truth in the temporal relationships. A series is used for this purpose, because the mechanism to record a past state transition can be reused.

Each specialist module presents values of plant variables in a future period as its proposing plant operation. It can be regarded as a scene in the future. Let us think of an arbitrator which is a kind of a specialist module containing a series to represent future state transition of a monitored entity. It manages the future series as a time table of the plant variables.

When a specialist module for guidance finds an plant operation, it sends a message to the arbitrator. The message is a requirement for registering a plant variable value in an associated future period. The arbitrator tries to register the plant variable value in the designated period. If the period is not already occupied by any other value, the register succeeds. Otherwise, the register fails, which means occurrence of a conflict.

Priorities has to be taken into consideration to solve a conflict. However, a method to determine a suitable priorities strongly depends on each problem. We have not find a general procedure to determine priorities, yet. The simplest method to solve a conflict is to make the specialist module trying the last register abandon its plant operation. Another method is explained in [49]

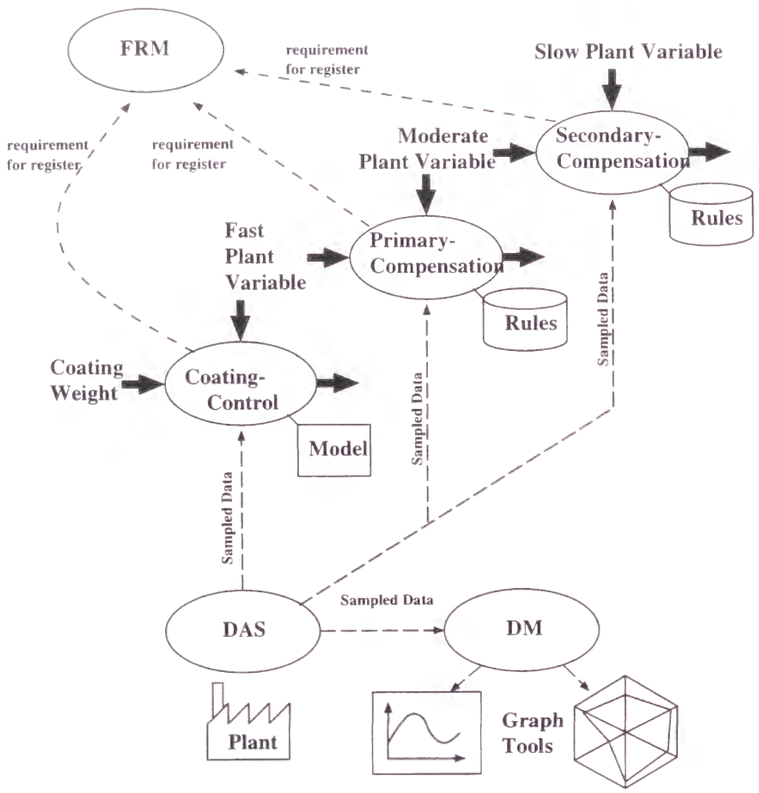



Figure 5.4: Plant Operation Supporting System

## 5.5 Evaluation in Actual Plant

### 5.5.1 System

The plant operation supporting system developed in this study consists of the components shown in figure 5.4 on a SONY NEWS 1750(NewsOS 3.3). The message-passing mechanism is implemented on Remote Procedure Call(RPC)[5].

The numbers of data items in aspects which `CoatingControl`, `PrimaryCompensation`, and `SecondaryCompensation` use to search plant operations are 6, 9, and 8, respectively. A cycle of the regular recognition of the internal series in each specialist module depends on the response time of the plant variables the specialist module treat as manipulated variables. `CoatingControl`, `PrimaryCompensation`, and `SecondaryCompensation` have to activate their methods every 10 seconds, every 30 seconds, and every 600 seconds, respectively.



A DAS in the system transfers some of 26 kinds of plant variable values sampled every 3 seconds with a time-stamp to Data Manager(DM), `CoatingControl`, `PrimaryCompensation`, and `SecondaryCompensation`. In DM, all plant variable values transferred from the DAS are stored in a series without conversion, while the three specialist modules convert the values into one in scenes of their own aspect. Graph tools, which are constructed on X window system, require DM for plant variable values in a specified period. Future Resource Manager(FRM) is a specialist module organizing plant operations proposed by the three specialist modules for guidances.

### 5.5.2 Knowledge

To suggest plant operations, each of the specialist modules has knowledge useful to accomplish the assigned task. The knowledge is represented by a set of rules in each specialist module. The condition part of a rule is represented by series skeletons. The guidance message in the action part is presented to an operator.

#### Mathematical Model

`CoatingControl` has several simplified mathematical models which are applicable to some of fast plant variables. It assumes that slow and moderate plant variables would have little effect on the coating weight, until an effect of modifying a fast plant variable appears. One of the models of `CoatingControl` is explained briefly.

Assume that consumed rate $r(mol/s)$ of coating metal $M$ depends on two kinds of fast plant variables: current density $e(A/m^2)$ and line speed $L(m/s)$. As it is shown in figure 5.6. the average coating weight per a unit area, $d(mg/m^2)$, is supposed to be proportionate to $e$; that is,

$$d = \varepsilon \cdot e, \tag{5.1}$$

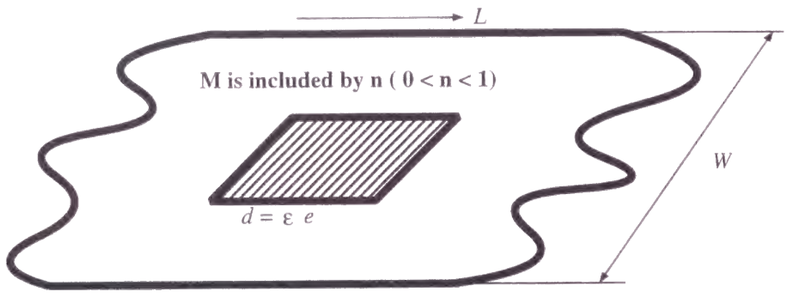



Figure 5.5: Steel Plate

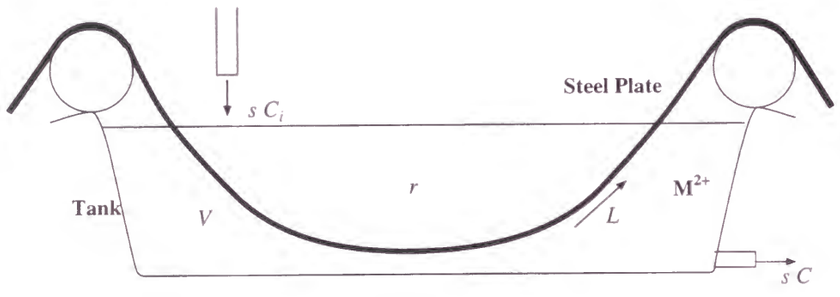



Figure 5.6: Tank for Galvanization

where $\varepsilon$ is a constant. $M$ is included in the whole coating substance by the proportion $n(0 \leq n \leq 1)$. Using the width of the steel plate, $w(m)$, $r$ is represented by

$$r = \varepsilon \cdot e \cdot (w \cdot L) \cdot n. \tag{5.2}$$

Suppose the concentration of the solution entering into the tank and the concentration of the solution in the tank are $C_i$ and $C$, respectively. In the tank, $M$ increases by $s \cdot C_i$ and decreases by $S \cdot C + r$. Therefore,

$$V \cdot \frac{dC}{dt} = s \cdot C_i - s \cdot C - r, \tag{5.3}$$

where $V(m^3)$ is the volume of the solution and $s(m^3/s)$ is the flow speed in the tank. Since $e$ and $L$ is far faster than $C$, $C$ is regarded as constant while $e$ and $L$ are changing. Thus,

$$r = s \cdot C_i - s \cdot C. \tag{5.4}$$



```
Rule1:
 if((the coating weight has been in the lower part of
the normal range for 600 seconds) &&
   (the present line speed is normal) &&
   (the present current density is normal) &&
   (the present temperature is higher
 than a normal upper limit))
 then take a plant operation:
   (raise the concentration).

Rule2:
 if((after an hour from increment of the concentration,
a feedback value of coating weight is
within the upper part of a normal range) &&
   (the present line speed is normal) &&
   (the present currency density is normal) &&
   (the present temperature is higher
 than a normal upper limit))
 then take a plant operation:
   (pull the temperature down).


[N.B.]
      To be easily understood, the periods and
      plant variables are represented by their meaning.
```

Figure 5.7: Example of Rules

From (5.2) and (5.4), a pair of $e$ and $L$ is computable because other variables are measurable.

In reality, `CoatingControl` contains a series of an aspect whose characteristic variable is computed from all fast plant variable values. `CoatingControl` regularly examines the series to determine whether modification is needed. Upon modification, `CoatingControl` selects a proper model with rules, and proposes a value of the plant variable calculated from the model.

### Compensation

When the feedback value of a fast plant variable is close to its limit, `PrimaryCompensation` searches a plant operation compensating it with a moderate plant variable. The specialist module expects a situation where the fast plant variable can be modified without undesirable

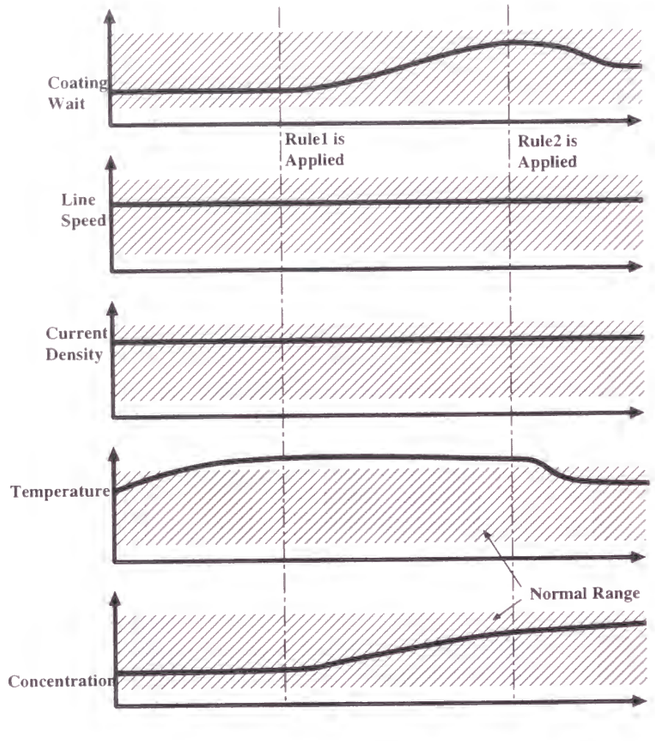



Figure 5.8: Application of Rules

effects. When an expected situation actually appears, it modifies the fast plant variable away from its limit. In same ways, `SecondaryCompensation` compensates moderate plant variables with slow plant variables. `PrimaryCompensation` and `SecondaryCompensation` have heuristic rules based on the operator's experiences. They propose plant operations only when specific conditions are satisfied.

For example, `SecondaryCompensation` has the rules shown in figure 5.7. Plant variables are generally compensated with a sequence of planned plant operations. The early parts of the charts in figure 5.8 illustrates a situation in which `Rule1` is applicable. The coating weight is expected to increase by the plant operation proposed in `Rule1`. When the coating weight increases as expected, `Rule2` is applied, for it would not deviate from the desirable range even if the temperature is reduced. As this example illustrates, the application of `Rule2` is planned when `Rule1` is applied.



The numbers of rules which are used in `CoatingControl`, `PrimaryCompensation`, and `SecondaryCompensation` are 11, 27, and 19, respectively. The rules are very few because of two features of this system. One is the specification of rules with series skeletons. The series skeletons enable the rules to concisely specify conditions for selection of moments to modify plant variables. The other is the classification of plant variables by their response time, accompanied with assignment of a clear role to every group of plant variables. The regulation of the coating weight is the role of the fast group, and the compensation is the role of the other groups. The second feature encapsulates rules for each group into one specialist module. Thus, the rules in one specialist module can be described with a little consideration of the rules in other specialist modules.

### 5.5.3 Comparison with Experienced Operator

The practicability of the developed plant operation supporting system has been verified in a real plant. The plant was operated by an experienced operator who could not see the plant operation supporting system. At the same time, the system suggests plant operations. We logged both plant operations. Later, the log of the system was compared with that of the operator.

In this experiment, three specialist modules proposed 539 plant operations in all. 39 plant operations of them indicate to modify specific plant variables. 7 plant operations of them are not identical with what the operator actually does. To the contrary, the experienced operator modifies some plant variables 4 times, while the plant operation supporting system do not suggest the modification of any plant variable.

The percent of the proper modification proposed by the plant operation supporting system is

$$(39 - 7)/39 \cdot 100 = 82.05.(\%)$$

The percent of the proper plant operations proposed by the plant operation supporting system was the following.

$$(539 - (7 + 4))/539 \cdot 100 = 97.96.(\%)$$

These figures prove that the plant operation supporting system is useful.

The plant operations which are not identical with what the operator actually does are suggested when a welded part of steel plates was passing in the tank. Different strategies are needed to regulate the coating weight in such a situation. Switching between strategies depending on situations is our future work.

## 5.6 Agents to Recognize Series

A plant operation supporting system is developed using a distributed AI method. The system reflects expertise an experienced operator has for the steel galvanizing plant. They regulate the coating weight with fast plant variables, while they use plant variables whose effects does not appear immediately for the compensation.

This chapter explains two kinds of agents for guidance. One agent proposes the modification of the fast plant variables to regulate the coating weight, based on rough mathematical models. The others suggest how to compensate faster plant variables with slower plant variables, based on heuristic rules. Matching of series with series skeletons are tried to examine trends. The experiment in a real plant has proved that the developed plant operation supporting system is useful.

Since trends are to be found, series are indispensable to represent the expertise. It is also noteworthy that the detection of trends needs not only the order of states of a monitored entity but also the length of intervals during which the entity stays in specific states. It is not addressed in conventional studies on temporal relationships[1][30][53].



# Chapter 6

# Example Expansion Maintaining Consistencies

## 6.1 Expertise for Control and Real-Time Computing

DAS systems for plants should maintain the timing consistency[40] which means any task completes data handling by a predefined deadline. The rate monotonic analysis(RMA)[26] is one of a few practical methods to predict whether the timing consistency is maintained. The RMA, however, cannot be applied unless tasks in a system have few interactions with each other. It also needs a precise estimation of the task execution time which varies with schemata and a hardware platform. The realization of DAS systems based on the RMA requires not only control techniques specific to each target problem but also programming techniques to construct a real-time system based on the RMA. Since there are various kinds of plants, no engineer is familiar with both of them. In general, DAS systems are constructed with the cooperation of control engineers who have control expertise for stable working of plants and real-time system engineers who have programming expertise for real-time computing.

To improve the stability of a plant, control engineers want to modify the configuration and schemata of a DAS system in the plant. It is not easy to modify the system without disrupting the timing consistency, because of huge program size. Although an approach to synthesize software components[58] is proposed for the reconfiguration of real-time systems,

numerous components are necessary to construct a practical system. Since the selection of components needs knowledge of them, the approach is not useful for control engineers unfamiliar with programming techniques. Components also contain much redundancy in terms of the reusability. It is difficult to know the precise execution time of tasks in a synthesized system, which makes it hard to apply the RMA.

The *Active*RING model[52] has been presented as a computational model for DAS systems. Here, the example expansion is proposed as a formal method to generate a target DAS system based on the *Active*RING model from requirements specified by control engineers. In the example expansion, the configuration and schemata in a small example are expanded, while an essential mechanism to maintain the timing consistency is preserved to a target system. The RMA is applied to the target system to guarantee its real-time behavior. Source codes of the target system are generated, because they enable the measurement of the execution time on arbitrary platforms. In this method, after defining a small schema which contains any data type used in the target system, real-time system engineers construct an example based on the schema in a small configuration. They specify rules to expand the configuration and schemata of the example according to the requirements of the control engineers. We have developed tools to expand the example into a target system automatically with the rules which preserve the predictability for maintenance of the timing consistency. The example expansion has the three following features.

- Expanded procedures are separated from a mechanism for the timing consistency.
- It maintains nested structures in expanded procedures.
- It maintains the dependency in statements constituting expanded procedures.

These features allow control engineers who have no knowledge of real-time computing to generate a DAS system. Real-time system engineers develop a mechanism to maintain the timing consistency and verify it on a small example. They can accomplish the work quickly. A small example also facilitates modifications. We have applied the example expansion to the development of actual DAS systems. A DAS system for a steel mill plant has been developed



with the example expansion 2.7 times faster than the time taken with a conventional method.

## 6.2 Customization of DAS System

### 6.2.1 Items for Customization

When a DAS system is customized to a specific application, the following items are generally customized for the application.

- schemata

  Since various clients works on their own purposes, Multifarious provision schemata are specified in each application. Acquisition schemata also vary with applications, because sensors and controllers are different in each applications.

- period for periodic acquisition

  Continuous data items are acquired in periods appropriate for them. Data are acquired in more than one periods.

- device for aperiodic acquisition

  For the periodic acquisition, input areas are implemented with memory mapped devices, which are accessed asynchronously. The access methods are similar in many devices. On the contrary, for the aperiodic acquisition, data are likely to be grouped for every input device. Several kinds of devices are used in an actual application. A specific access method should be prepared for each of them.

- ECA rules

  State Change to be detected are different in each application. A wide variety of rules are specified in a DAS system.

### 6.2.2 Customized Procedures in DAS

Let us consider to construct a DAS system for each application, based on the *Active*RING model. Some procedures in DAS system are customized, to accommodate the items enumerated above.

#### Schemata

First, provision schemata should be considered, because they are most abundant in their variety. More than one provision objects are prepared so that requests with different priorities can be processed simultaneously. Each provision object has methods specific to provision schemata in it. Needless to say, procedures implementing the methods depend on provision schemata.

There is a single chief object in a system based on the *Active*RING model. It assigns a request to an appropriate provision object according to provision schemata specified in the request. The customization of provision schemata affects on procedures of the chief object in the *Active*RING model.

In the *Active*RING model, a pair of a circular area and an acquisition object are associated with a specific acquisition schema. It is obvious that procedures in an acquisition object depend on an acquisition schema. Many devices used for the acquisition also affect procedures in an acquisition object.

#### Configuration

Periods and devices for acquisition affects on the configuration of a DAS system base on the *Active*RING model. In each acquisition object, an event activates a method to acquire data item values based on an acquisition schema. Events for periodic acquisition are sent from a timer, while arrival of data from a device to an input queue is regarded as an event to activate aperiodic acquisition. The pair of a circular area and an acquisition object is



prepared for each activation event: for each acquisition period or each device.

As acquisition periods affect the number of acquisition objects, the number of provision objects varies with periods in the periodic service. The configuration of a DAS system should have many provision objects if clients may specify various periods.

#### ECA Rules

In the *Active*RING model, ECA rules are divided into two categories: the on-acquisition conditions, and others.

The former are assumed to be represented by means of linear inequations with the form of (4.2). In (4.2), each data item is not referred to by name, but by an offset from the top address of the scene. The interpreter of the linear inequations is independent from schemata. It can be used in any DAS system.

ECA rules other than the on-acquisition conditions are dependent on provision schemata. They are executed in procedures in provision objects.

### 6.2.3 Dependent and Independent Procedures

Figure 6.1 shows the outlines of procedures in each object. The procedures dependent on each application are ones enclosed by rounded rectangles in figure 6.1.

The role of the chief object is to assign a request to an appropriate provision object. Since each provision object has methods to treat specific provision schemata, the procedure in the chief object is fully dependent on provision schemata and the configuration of the provision objects. A request is received in the top of the procedure, which is succeeded by the assignment of the request. The assignment is realized with a list of `if` statements to determine a provision object to accomplish the request. Note that the chief object has no exclusive information shared with other objects, though its procedure depends on provision schemata and the configuration of the provision objects.

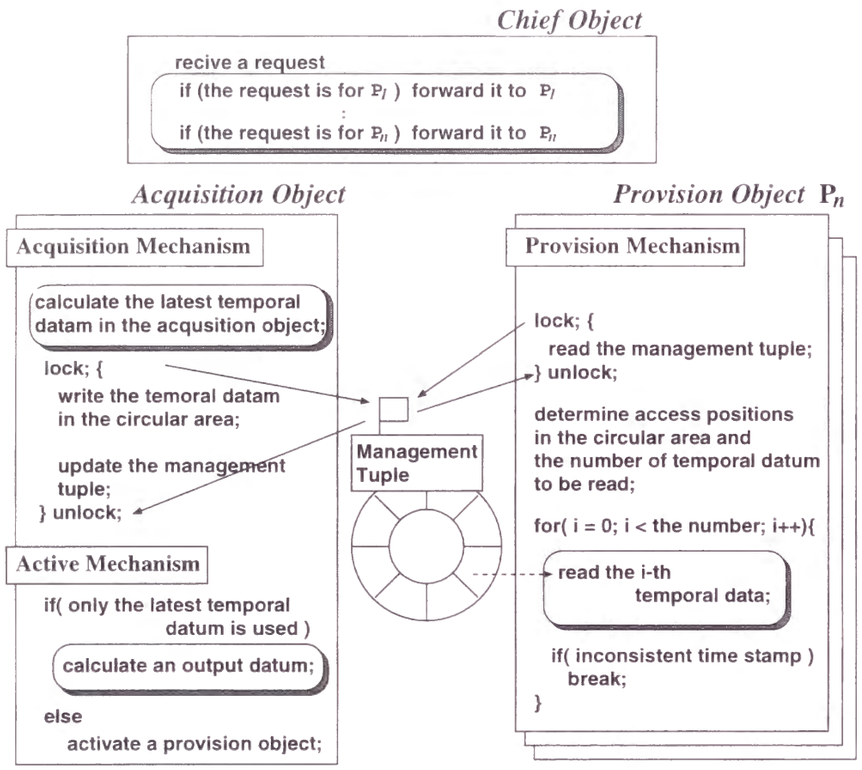



Figure 6.1: Procedures in *Active*RING

The outlines of procedures of in the acquisition objects and the provision objects are shown in figure 6.1. After the calculation of the latest temporal datum, an acquisition object writes the datum into a circular area, excluding for other objects to access to the management tuple. A provision object excludes accesses by the acquisition object while it reads the management tuple to know which part of the circular area is unlikely to be overwritten. The provision object makes access to the part without exclusion. In both case, procedures related to the exclusion are independent from each application. They are fixed in any DAS systems.

### 6.2.4 Applicability of RMA

As known from equation (2.4) in section 2.3.1, the RMA can be applied to know whether a DAS system maintains the timing consistency, if



1. we can measure the worst case execution time of each task when the task is solely executed, and

2. for each task, we can bound the blocking time caused by tasks of lower priority.

First, the worst case execution time of an object method in the *Active*RING model can be measured by executing it without involving any other task. Next, we will proceed the blocking time. It is useless to repeat provision of the latest datum before it is updated by acquisition, which means there can be no provision task more frequent than the acquisition. In the *Active*RING model based on the RMA, an acquisition object has a higher priority than a provision object. Its blocking time caused by the provision object is small, because the provision object locks only the management tuple instead of the whole circular area. In addition to that, it can be bounded because of the mutual exclusion mechanism based on the priority inheritance protocol. Therefore, we can apply the RMA to a DAS system based on the *Active*RING model to know whether the time constraints are satisfied.

## 6.3 Example Expansion

### 6.3.1 Generator Generator for Reconfiguration

We want to get customized DAS systems with little effort. The RMA should be applied to a reconfigured system so that the timing consistency is also maintained after the reconfiguration. The precise execution time is required for application of the RMA.

For precise estimation of the execution time, the execution time must be bounded, and the difference of the actual execution time from the bound should be small enough. Since specific devices are used to acquire values of data items in schemata which vary with each problem, it is effective to generate source codes for a target system. The precise execution time can be measured with the source codes compiled and executed on a target hardware platform.

The example expansion is proposed to develop a large scale target system from a small

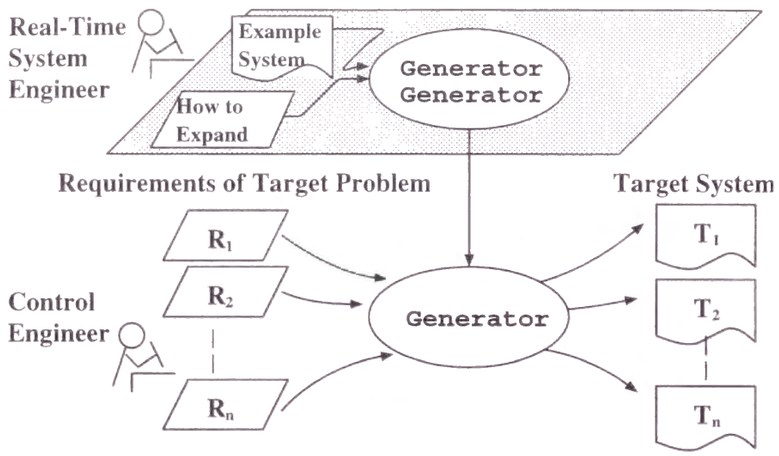



Figure 6.2: Generator Generator

example quickly. The example expansion is motivated by the fact that many system engineers construct new real-time systems, following mechanisms and techniques of working systems. In the example expansion, real-time system engineers are responsible for constructing an example system based on the *Active*RING model, so that it should work maintaining the timing consistency. The example system is founded on small schemata containing any kind of data type which can be used in a target system. It should be built in a small configuration to cover all functions which may be used in a target system. The example expansion generates a target system by expanding schemata and the object configuration.

The target system should be reconfigured by control engineers, without bothering real-time system engineers. We provide a generator generator which produces a generator from source codes of an example system and information specifying how to expand the source codes, as illustrated in figure 6.2. The generator produces a target system specific to each set of target requirements specified by control engineers. In this paper, all source codes are assumed to be written in C language, which is used in the development of most real-time systems.



### 6.3.2 Example System

To generate a target system, procedures for the configuration and schemata in an example system are expanded according to the requirements of the target problem. The procedures correspond to the codes enclosed by rounded rectangles depicted in figure 6.1. The enclosed codes in acquisition objects and provision objects, which depend on schemata, have no relationships with codes for the mutual exclusion. The expansion of the object configuration affects the methods of the chief object which assigns a request to an appropriate provision object. Once the chief object assigns a request, it interferes with neither the acquisition objects nor the provision objects. The expanded procedures in the *Active*RING model are separated from essential codes to maintain the timing consistency. The expanded procedures should be specified in the following manner so that the example expansion may preserve the predictability for maintenance of the timing consistency in a target system.

The expanded procedure in the chief object determines a provision object the request is forwarded to. The procedure is implemented with a list of `if` statements as shown in figure 6.1. By expanding the list, it can accommodate the customization of the provision object configuration.

The procedures in acquisition objects and provision objects implement the value assignment to each data item in a schema. The expansion of the procedures founds on increase of data items in schemata. Figure 6.3 shows a simplified procedure to acquire values in an acquisition schema. An example system is supposed to call the procedure for every acquisition. The left side of the figure shows line numbers. In this procedure, whole data on an input area are read to a local variable `area`, which is used to set values of base data items `alpha` and `beta`. Derived data item `max` takes the larger one of them as its value, while `min` takes the smaller one. The procedure consists of several components. The lines from $l_7$ to $l_9$ constitute a component to calculate the value of a base data item. In the component, function `conversion` changes a datum specified with `offset` and `length` to the value of base data item `alpha`. On the other hand, the lines from $l_{13}$ to $l_{17}$ constitute a component to calculate the value of the derived data item. The component for a base data item specifies

```
l1:  void write_RB(sp)                                          b_hd
l2:     struct _scene_RBschema *sp;
l3:  {
l4:     char area[INPUT_AREA_SIZE];
l5:     int offset, length;
                                                                b_prl
                    /* Read input area data*/                   b_init
l6:     read_RB_inputArea(&area, INPUT_AREA_SIZE);
                                                                b_bdy
                    /* Calculate a base data item */            b_bsc
l7:     offset = 8;  /* offset in input area */
l8:     length = 2;  /* length of input data */
l9:     conversion(&(sp->alpha), area + offset, length);
l10:    offset = 16;
l11:    length = 1;
l12:    conversion(&(sp->beta), area + offset, length);
                    /* Calculate a derived data item */         b_drv
l13:    maximum(&(sp->max),
l14:            &(sp->alpha),
l15:            &(sp->beta) );
                                                                b_avs
l16:    minimum(&(sp->min), &(sp->alpha), &(sp->beta));
                                                                b_bdy
l17: }                                                          b_tl
                                                                b_whl
```

Figure 6.3: Example Procedure for Acquisition

the conversion from an input area to an acquisition schema, while that for a derived data item specifies conversion from acquired data item values. In the same way, each procedure in the active mechanism and the provision mechanism specifies conversion of a data item value in a provision schema from data item values in acquisition schemata. The conversion is implemented with a function which calculates an output from inputs with only local variables. A set of functions are prepared for each target problem.



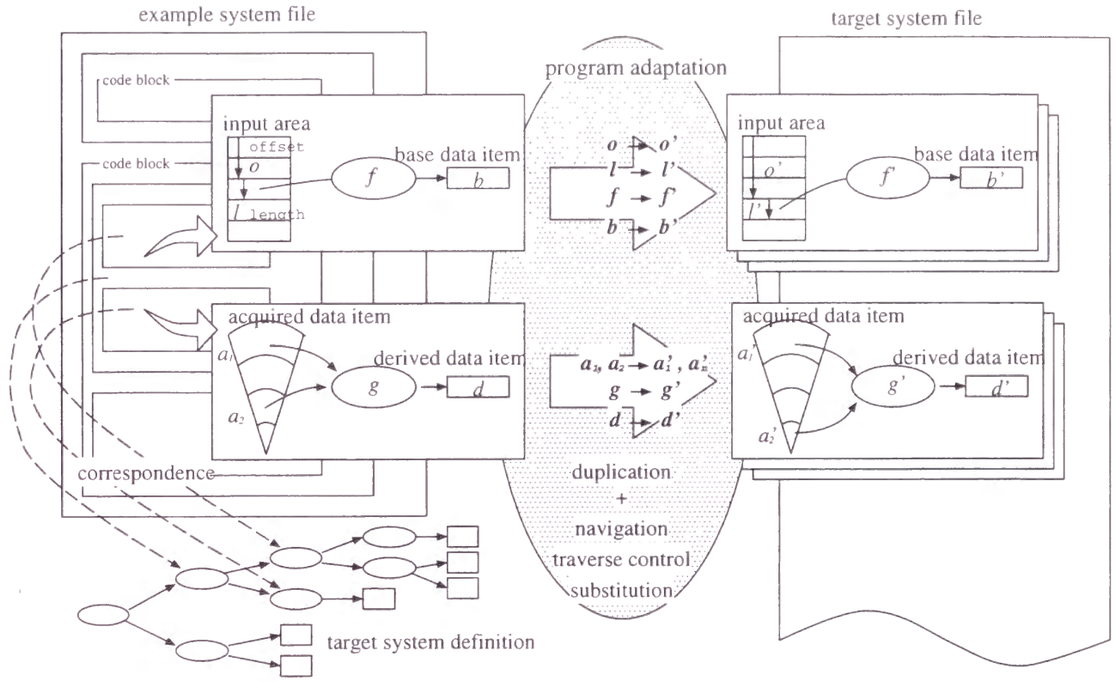



Figure 6.4: Basic Idea of Example Expansion

### 6.3.3 Basic Idea For Expansion

The chief object methods are implemented with the reception of a request R and its assignment to provision object P. The simplest source codes are

```
receive request R;
if(R is for P)  forward R to P;
```

The expansion of the methods can be realized with the iteration of the second line along with substitution of string P.

The basic idea to expand an acquisition schema is illustrated with the simplified procedure shown in figure 6.3. To produce source codes to calculate a specific data time, we duplicate a component selected according to the kind of the data item and substitute appropriate strings in the component, as illustrated in figure 6.4. Strings to present the offset, the data length,

the conversion function, and the target data item are substituted for a base data item, while strings for the data items used in the derivation, the conversion function, and the target data item are substituted for a derived data item. The iteration of the duplication along with the substitution according to a given schema enables us to produce codes for calculation of data items in the schema.

The basic idea is to iteratively or selectively duplicate components along with string substitution according to the requirements of a target problem. In addition to that, the correctness of procedures in an example system must be transferred to a target system. The items indispensable for the example expansion are:

- components of procedures,
- requirements of a target problem,
- rules specifying iteration times, selection conditions, and substitution,
- transfer of the program correctness.

Each of the items is explained in the succeeding sections.

### 6.3.4 Code Blocks

Components of procedures implementing an example system are used as bases of target system procedures. As a component in source codes, we consider a code block which matches one of the following:

- maximum sequence of lines which are used without division,
- a sequence of code blocks which is used 0 or more times, and
- a sequence of code blocks, at most one of which is selected to be used.

We represent code block $b_p$ consisting of line $l_i, ...l_j$ to appear successively in an object method with notation $b_p(l_1, ..., l_m)$. Notation $b_p(b_1, ..., b_k)$ means $b_p$ is a sequence of $b_1, ..., b_k$.



In some code blocks, specific strings are substituted, while others are used as they are. The former are referred to as *adaptable*, and the latter are as *rigid*. An *atomic* code block includes no other code blocks. We classify code blocks in the hierarchy.

**part** : a code block which is a sequence of rigid cells and adaptable cells.

  **rigid cell** : an atomic and rigid code block.

  **adaptable cell** : a sequence of cytoplasm and nucleus which may be used iteratively.

    **cytoplasm** : an atomic and adaptable code block.

    **nucleus** : a code block which is a sequence of genes and parts. Either of them is selected in a nucleus.

      **gene** : an atomic and adaptable code block used selectively.

      **part**

The whole procedure is regarded as one part. Since a nucleus can include parts, a whole procedure is represented by a nested structure of code blocks.

In the example expansion, a code block is duplicated with its strings substituted. The order of components is maintained in the code block. Note that a sequence of components in an adaptable cell can be iteratively duplicated, and at most one of the components is selected in a nucleus. Since code blocks are recursively expanded, the example expansion transfers a nested structure and sequences in a procedure of an example system to a target system.

### 6.3.5 Requirements in Target System

Schemata and an object configuration in a target system are referred to as a *target system definition*. Procedures in an example system are expanded according to a target system definition. A target system definition includes the following information for each kind of object.

**chief object** pairs of a request and a provision object to process it.

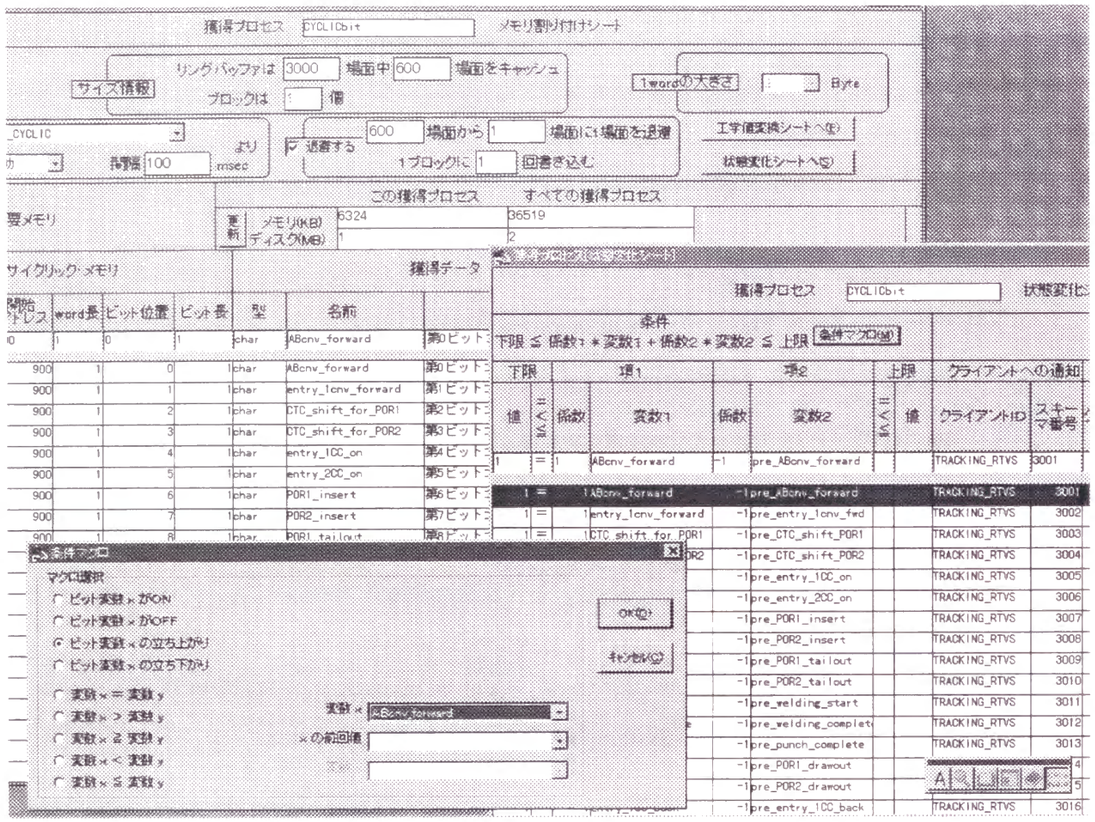



Figure 6.5: Builder

**acquisition object** data items in an acquisition schema along with information to set their values, and ECA rules.

**provision object** data items in a provision schema along with information to set their values.

We have developed Servish, a language specialized to specify a real-time control system based on the *Active*RING model. As shown in figure 6.5, a builder which has a spread sheet like GUI has been developed, so that a target system definition may be specified by control engineers who are not familiar with programming. The output of the builder is a Servish



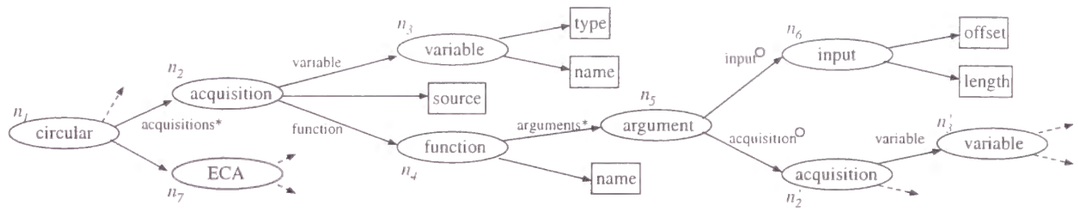



Figure 6.6: Subtree in Target System Definition

program. The parser of Servish converts a Servish program into a tree, like the one shown in figure 6.6. In the figure, a vertex represented by a rectangle denotes a terminal which has a literal value, while a vertex represented by an oval denotes a non-terminal composed of one or more members. A label associated with an arc is the member name with which the source vertex refers to the destination vertex. An asterisk following a label on an arc means that the source of the arc refers to the destination more than 0 times. On the other hand, a circle following a label means that the source selects one of the destinations.

### 6.3.6 Expansion Rules

In this section, an expansion rule is defined after notations needed for the definition.

Notation $l_i \Leftarrow n_j$ means that strings of line $l_i$ are substituted according to the information that vertex $n_j$ has in a target system definition. *Substitution* $R$, having taken place in a code block, is represented as $R = \{l_i \Leftarrow n_k, ..., l_j \Leftarrow n_l\}$. When no string is substituted, the substitution is $\phi$.

Code blocks are traversed to transfer a nest structure and sequences in a procedure in an example system to a target system. A *traverse control* specifies an instruction to traverse code blocks in a manner other than the depth-first manner. Traverse control $T$ is one of $itr(n.m)$, $sel(n.m)$, and $case(l)$, where $m$ is a member of non-terminal vertex $n$ and $l$ is a literal in a target system definition. In the example expansion, code blocks in an adaptable cell can be used iteratively. The number of iterated times is specified with notation $itr(n.m)$. The code block is used as many times as the reference number of $m$ in $n$. On the other hand, a

code block can be used selectively. The selection is specified with a combination of a nucleus and its component code blocks, that is, genes and parts. Suppose vertex $n$ corresponds to a nucleus code block. A combination of notation $sel(n.m)$ specified together with the nucleus and notation $case(l)$ with its component indicates that the component is selected in the nucleus if the value of member $m$ of vertex $n$ is equal to literal $l$.

Iteration times, selection conditions, and substitutions are specified with a vertex in a target system definition. Each code block should be traversed identifying the vertex corresponding to it. Suppose code block $b$ corresponds to vertex $n_e$. Let $n_{r_1}, ..., n_{r_k}$ be necessary for either substitution in $b$ or traverse to a code block which is a component of $b$. Let navigation $L(n_{r_m})$ be a path such that it consists of arcs from $n_e$ to $\forall\ n_{r_m} \in \{n_{r_1}, ..., n_{r_k}\}$. A *navigation set* is

$$\bigcup_{1 \le m \le k} L(n_{r_m}), \tag{6.1}$$

which represents a set of arcs used in the vertex reference in code block $b$. Notation $n_d \leftarrow n_s$ is used to represent the fact that there is an arc from $n_s$ to $n_d$. Navigation set $E$ is represented as $E = \{n_r \leftarrow n_i, n_i \leftarrow n_j, ..., n_k \leftarrow n_e\}$.

An expansion rule is defined so as to be associated with correspondence from a code block to a vertex. Suppose code block $b_p$ which consists of code block $b_{c_1}$, ..., $b_{c_m}$ corresponds to vertex $n_e$ in a target system definition. Suppose $b_{c_1}, ..., b_{c_m}$ corresponds to vertex $n_1, ...., n_m$, respectively. An *expansion rule* is a triple of navigation set $E$, traverse control $T$, and substitution $R$, such that

- $E$ includes all arcs in a navigation from $n_e$ to any element of $\{n_1, ...., n_m\}$,
- $E$ includes all arcs in a navigation from $n_e$ to any vertex used in the traverse control, and
- $E$ includes all arcs in a navigation from $n_e$ to any vertex used in any element of $R$.

Notation $b_p(l_i, ...l_j) \stackrel{\alpha_p}{\longmapsto} n_e$ means that code block $b_p$ consisting of line $l_i, ...l_j$ corresponds to vertex $n_e$ with expansion rule $\alpha_p$.



### 6.3.7 Maintaining Correctness

To transfer the program correctness from an example system to a target system as well as the timing consistency, we must pay attention to *control dependences* and the *data dependences*[11][13].

**control dependence** Let expression $e$ be a condition part of a conditional statement, say `if-then-else`, or a loop statement, say `while`. If evaluation of $e$ determines whether statement $s$ is executed, there is a control dependence. Statement $s$ depends on $e$.

**data dependence** Suppose a program where variable $v$ is defined in statement $s$ and referred to in statement $t$. When the program has an execution path from $s$ to $t$ with no definition of $v$, there is a data dependence. Statement $t$ depends on $s$ with regard to $v$.

Real-time system engineers should transfer control dependences by including components of a conditional statement or a loop statement in a single code block. Let statement $s$ in code block $b$ define variable $v$. Suppose $v$ is referred to in statement $t$ in code block $c$. Statement $t$ depends on $s$. When $v$ is to be substituted with $u$, the data dependence with regard to $v$ must be transferred to a target system. Real-time system engineers should specify expansion rules so that $v$ should be substituted with $u$ in both $s$ and $t$, maintaining the order of $b$ and $c$. In addition to that, neither mistake nor conflict is allowed in substituted variable names. For example, a target system definition where a control engineer mistakes the name of an acquired base data item in derivation of another data item cannot produce a correct target system. The semantic check mechanism of the Servish parser finds mistakes and conflicts in variable names.

Through many tests, real-time system engineers confirm correct behaviors of an example system. To get a target system, methods in the example system is expanded with the expansion rules which maintain both of the data dependence and the control dependence. The target system founds on the *Active*RING model as the example system does. The worst case execution time of every method in the target system can be measured, by executing

the method solely. In the *Active*Ring model, expanded codes are separated from the codes for the mutual exclusion. We can know the worst case blocking time of each task. Using the RMA, it can be predicted whether the timing consistency is maintained in the target system.

### 6.3.8 Illustrations of Expansion Rules

The subtree depicted in figure 6.6 shows how to set the values of data items. Vertex $n_1$ and $n_2$ correspond to a circular area and acquisition of each data item, respectively. Since a circular area has multiple data items, the arc from $n_1$ to $n_2$ is labeled with `acquisition*`. In acquisition of each data item, the value calculated with function $n_4$ is set to data item $n_3$. When member `source` of $n_2$ takes `BASIC` as its value, the data item to be acquired is a base one. Then, a value on an input area indicated by $n_6$ is an argument of function $n_4$. When it takes `DERIVED`, the data item is a derived one. Arguments of function $n_4$ are represented by $n'_2$ and $n'_3$[1], data item values which have been acquired.

Let us expand the procedure shown in figure 6.3 according to the subtree in the target system definition. In code block $b_{bdy}$, $b_{avs}$ is used as many times as the value represented by `acquisitions*` in $n_2$, to produce source codes for the acquisition of all specified data items. It is represented by the traverse control in

$$
\begin{aligned}
& b_{bdy}(b_{avs}) \stackrel{\alpha_{bdy}}{\longmapsto} n_1, \\
& \alpha_{bdy}(E_{bdy}, itr(n_1.acquisitions), \phi), \\
& E_{bdy} = \{n_2 \leftarrow n_1\}.
\end{aligned}
\tag{6.2}
$$

Code block $b_{bdy}$ and $b_{avs}$ correspond to vertex $n_1$ and $n_2$, respectively. Navigation from $n_1$ to $n_2$ is necessary when the traverse point moves from $b_{bdy}$ to $b_{avs}$. Navigation set $E_{bdy}$ in expansion rule $\alpha_{bdy}$ includes the navigation. In code block $b_{avs}$, either $b_{bsc}$ or $b_{drv}$ is selected depending on whether the data item to be acquired is a base one or a derived one. The expansion rule for $b_{avs}$,

$$
\alpha_{avs}(\phi, sel(n_2.source), \phi),
\tag{6.3}
$$

[1] $n'_2$ and $n'_3$ are different vertices of the same data types as $n_2$ and $n_3$, respectively



means that member `source` of vertex $n_2$ is referred for the selection. Code block $b_{bsc}$ to calculate a base data item corresponds to vertex $n_2$ with expansion rule

$$
\begin{aligned}
& \alpha_{bsc}(E_{bsc}, case(BASIC), R_{bsc}), \\
& E_{bsc} = \{n_3 \leftarrow n_2, n_4 \leftarrow n_2, n_5 \leftarrow n_4, n_6 \leftarrow n_5\}, \\
& R_{bsc} = \{l_7 \Leftarrow n_6, l_8 \Leftarrow n_6, l_9 \Leftarrow n_4, n_3, n_6\}.
\end{aligned}
\tag{6.4}
$$

On the other hand, $b_{drv}$ to calculate a derived data item corresponds to vertex $n_2$ with expansion rule

$$
\begin{aligned}
& \alpha_{drv}(E_{drv}, case(DERIVED), R_{drv}), \\
& E_{drv} = \{n_3 \leftarrow n_2, n_4 \leftarrow n_2, n_5 \leftarrow n_4, n'_2 \leftarrow n_5, n'_3 \leftarrow n'_2\}, \\
& R_{drv} = \{l_{13} \Leftarrow n_4, n_3, l_{14} \Leftarrow n'_3, l_{15} \Leftarrow n'_3\}
\end{aligned}
\tag{6.5}
$$

In the rules, substitution $R_{bsc}$ and $R_{drv}$ need navigation set $E_{bsc}$ and $E_{bsc}$, respectively. Each of $\alpha_{bsc}$ and $\alpha_{drv}$ has the traverse control in the form of $case(l)$ to indicate the code block is selected when the value of `source` is equal to $l$.

## 6.4 Tools for Example Expansion

In the example expansion, real-time system engineers identify code blocks in procedures to be expanded, and state expansion rules for the code blocks. Tools are prepared to support the example expansion. The generator generator, an essential tool, produces source codes of a generator for target systems as depicted in figure 6.2.

### 6.4.1 Tags

For the example expansion, real-time system engineers represent a expanded procedure with a nested structure of code blocks. They also have to specify an expansion rule for each correspondence of a code block to a vertex in a target system definition. The expansion rule also states the control of the code block traverse, and the string substitution. Tags are provided for real-time system engineers to specify a code block and an expansion rule. The

| | starting tag | | | | | | | | | | inside text | ending tag | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | external control label | type | role | arg. | param. | local variable list | internal control label | conv. list | | | | type | |
| rigid cell | /* ~!^ | | rgd | <*role*> | () | { } | { } | | () | ( */ | | /* )~!^ | rgd | */ |
| adaptable cell | /* ~!^ | | adp | <*role*> | (*arg*) | {*type param;*} | {*type var=exp;* ...; *type var=exp;*} | for(*exp* ; *exp* ; *exp* ) | () | ( */ | | /* )~!^ | adp | */ |
| cytoplazm | /* ~!^ | | cyt | <*role*> | (*arg*) | {*type param;*} | {*type var=exp;* ...; *type var=exp;*} | | (*conv*, ..., *conv*) | ( */ | ..... .....~!^%( *cnvd*) .....~!^%s( *cnvd*) ..... | /* )~!^ | cyt | */ |
| nucleus | /* ~!^ | | ncl | <*role*> | (*arg*) | {*type param;*} | {*type var=exp;* ...; *type var=exp;*} | *switch(exp)* | () | ( */ | | /* )~!^ | ncl | */ |
| gene | /* ~!^ | case(val) | gen | <*role*> | (*arg*) | {*type param;*} | {*type var=exp;* ...; *type var=exp;*} | | (*conv*, ..., *conv*) | ( */ | ..... .....~!^%( *cnvd*) .....~!^%s( *cnvd*) ..... | /* )~!^ | gen | */ |
| part | /* ~!^ | case(val) | prt | <*role*> | (*arg*) | {*type param;*} | {*type var=exp;* ...; *type var=exp;*} | | () | ( */ | | /* )~!^ | prt | */ |

Figure 6.7: Syntax of Tags

syntax of tags is shown in figure 6.7. Source codes of a procedure to be expanded are tagged like figure 6.8 whose original is shown in figure 6.3.

There are six kinds of code blocks. For each kind of code blocks, a pair of a starting tag and an ending tag is provided to specify a code block, enclosing successive source code lines. Starting tags begin at string "`/* ~!^`" and end with string "`( */`", while ending tags are concatenations of string "`/* )~!^`", the kind of the code block, and string "`*/`".

The starting tag consists of the following elements which indicate the information necessary for the example expansion:

- the kind which is one of `rgd`, `adp`, `cyt`, `ncl`, `gen`, and `prt`,
- the role which gives the code block an identifier unique in its parent code block,
- the argument and the parameter which shows the correspondence of the code block to a vertex, and



```
                    /* ~!^prt<whole>(circular){struct circular *circular;}{}() ( */
                     /* ~!^adp<prologue>(circular){struct circular *circular;}{}() ( */
                      /* ~!^cyt<header>(circular){struct circular *circular;}
                         {struct schema *schema = point_schema(circular->schema);}
                         (circular->name, schema->name) ( */
void write_~!^%s(RB)(sp)
  struct _scene_~!^%s(RBschema)  *sp;
{
    char area[INPUT_AREA_SIZE];
    int offset, length;
                        /* )~!^cyt */
                        /* ~!^cyt<initialization>(circular){struct circular *circular;}
                           {}(circular->name) ( */
    /* Read input area data*/
    read_~!^%s(RB)_inputArea(&area, INPUT_AREA_SIZE);
                        /* )~!^cyt */
                       /* )~!^adp */
                       /* ~!^adp<body>(circular){struct circular *circular;}
                          {struct acquisition *acquisition;
                           int count = circular->num_acquisition;}
                          for(acquisition = circular->acquisitions;
                              count--;
                              acquisition = point_acquisition(acquisition->next)) () ( */
                        /* ~!^ncl<data item>(acquisition){struct acquisition *acquisition;}{}
                           switch(acquisition->source) () ( */
                         /* ~!^case(BASIC) gen<base data item>(acquisition)
                            {struct acquisition *acquisition;}
                            {struct variable *var = point_variable(acquisition->variable);
                             struct function *fnc = point_function(acquisition->function);
                             struct argument *arg = point_argument(fnc->arguments);
                             struct input *input = point_input(arg->input);}
                            (input->offset, input->length, fnc->name, var->name) ( */
    /* Calculate a base data item */
    offset = ~!^%d(8);                    /* offset in input area */
    length = ~!^%d(2);                    /* length of input data */
    ~!^%s(conversion)(&(sp->~!^%s(alpha)),  area + offset, length);
                      /* )~!^gen */
    offset = 16;
    length = 1;
    conversion(&(sp->beta),  area + offset, length);
                     /* ~!^case(DERIVED) gen<derived data item>(acquisition)
                        {struct acquisition *acquisition;}
                        {struct variable *var = point_variable(acquisition->variable);
                         struct function *fnc = point_function(acquisition->function);
                         struct argument *arg = point_argument(fnc->arguments);
                         struct acquisition *acquired = point_acquisition(arg->acquisition);
                         struct variable *base1 = point_variable(acquired->variable);
                         struct variable *base2 = point_variable(base1->next);}
                         (fnc->name, var->name, base1->name, base2->name) ( */
     /* Calculate a derived data item */
    ~!^%s(maximum)(&(sp->~!^%s(max)),
                   &(sp->~!^%s(alpha)),
                   &(sp->~!^%s(beta))  );
                    /* )~!^gen */
    minimum(&(sp->min),  &(sp->alpha),  &(sp->beta));
                     /* )~!^ncl */
                    /* )~!^adp */
                    /* ~!^rgd<tailer>(){}{}()( */
}
                    /* )~!^rgd */
                   /* )~!^prt */
```

Figure 6.8: Tagged File

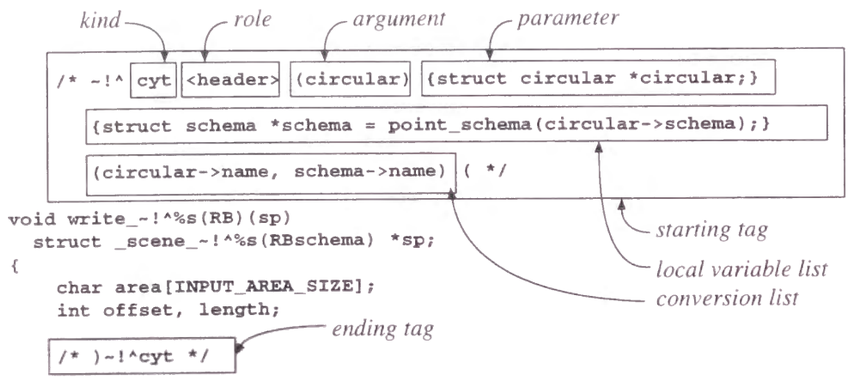



Figure 6.9: Elements in Tag

- the elements for the expansion rule such as
  - the local variable list for the navigation set,
  - the control label for the traverse control, and
  - the conversion list for the substitution.

Figure 6.9 illustrates the elements in the tag for code block $b_{hd}$ in figure 6.3. Both of the argument and the parameter indicates an identical vertex which corresponds to the code block. The data type specified along with the parameter is the one of the vertex. In the local variable list, a local variable and its data type are declared. A value may be assigned to a local variable, if necessary. The assignment means to navigate a link from one vertex to another in a target system definition. The assignment sometimes needs supplementary subroutines based on semantics of the vertices. Real-time system engineer can define semantic functions in C language for the assignment. The internal control labels are specified in starting tags for adaptable cells and nucleuses. The control label in an adaptable cell means *itr(member)*. Its form is "`for(expression; expression; expression)`", which has the same semantics of the `for`-statement in C language. The meaning of the control label in a nucleus corresponds to *sel(member)*. The control label has form "`switch(expression)`". The external control labels are specified in starting tags for genes and parts. In both of them, the control labels mean *case(literal)*.



The conversion list appears in starting tags for cytoplasm and genes. In these code blocks, strings to be substituted are marked with either

- a pair of "`~!^%s(`" and "`)`", or
- a pair of "`~!^%d(`" and "`)`".

The $i-$th marked string is substituted with the value of the $i-$th element in the conversion list. In the substitution, the element is evaluated as a string for the former pair, while it is evaluated as an integer for the latter pair.

### 6.4.2 Work Flow

Figure 6.10 depicts the tools for the example expansion. Control engineers create a Servish program which specifies a configuration and schemata in a target system using the builder. The Servish parser converts the program into a target system definition. On the contrary, real-time system engineers are responsible for building a correct example system based on the *Active*RING model. Some of procedures implementing it should be expanded for a target system, and others remain unchanged. Real-time system engineers attach tags to procedures to be expanded. The process of the example expansion is divided into two steps: the interpretation of tags and the generation of a target system.

In the interpretation of tags, g_g, which stands for a generator generator, produces generator procedures, template procedures, and substitution rules.

**generator procedures** A starting tag attached to a code block is converted into a subroutine which expands the code block. The argument and the parameter in a starting tag are used in the subroutine call and in the subroutine definition, respectively. In an adaptable cell tagged with a control label for the iteration, g_g produces a `for`-statement to iteratively call subroutines which expand its child code blocks. In a nucleus tagged with a control label for the selection condition, g_g produces a `switch`-statement, which contains `case`-labels corresponding to control labels attached to child

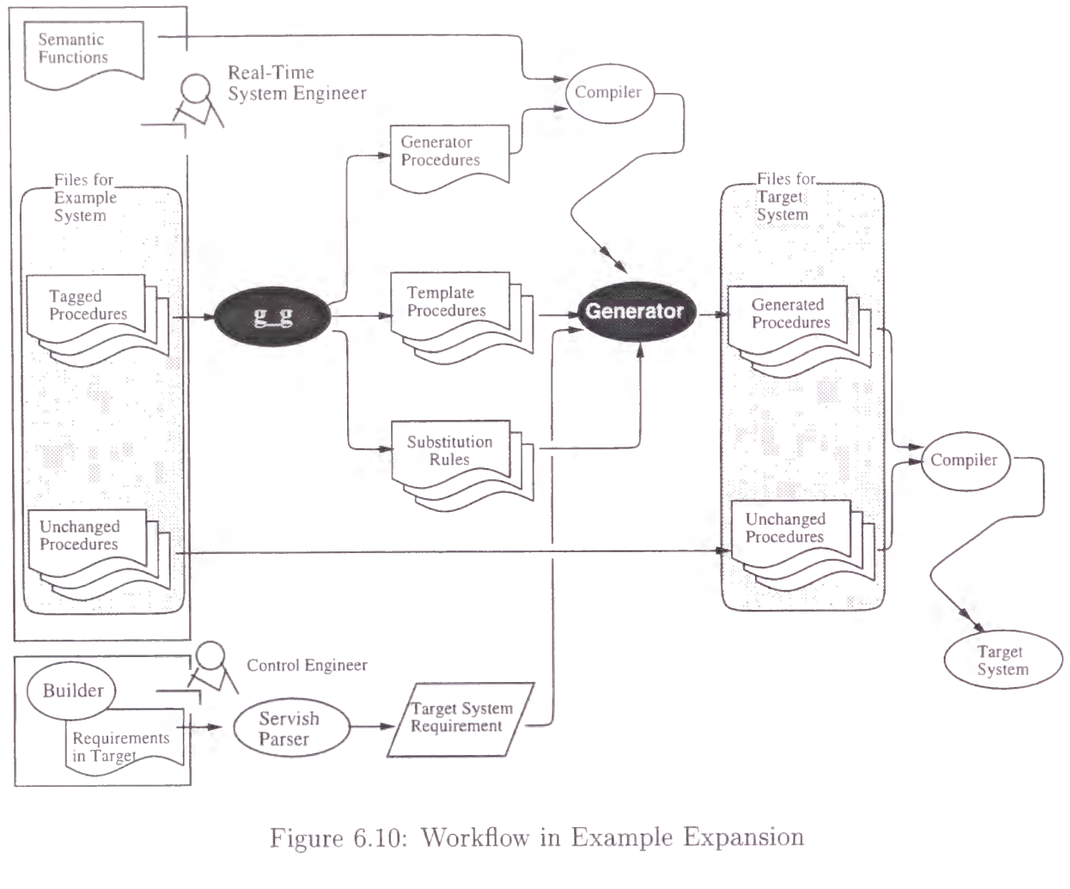



Figure 6.10: Workflow in Example Expansion

code blocks of the nucleus. The logic implemented in a subroutine is so simple that almost subroutines in a generator consist of less than a decade of lines.

**template procedures** Template procedures are produced by duplicating tagged procedures except the following two points. For each starting tag, g_g leaves only its kind and role enclosed with "`/* ~!^`" and "`( */`". Strings marked to be substituted in code blocks are returned to original ones. Template procedures make the nested structure of code blocks explicit. Produced template procedures can be compiled as a part of the example system, because any C compiler regards left tags as comments.

**substitution rules** Cytoplasm and genes extracted from tagged procedures are recorded as substitution rules, after the position of substituted string is clarified.



In the generation of a target system, a produced generator traverses code blocks, navigating arcs in a target system definition. The traverse is realized by the subroutine calls. In each subroutine, the generator calls subroutines corresponding to the child code blocks based on the nested structure, unless there is a traverse control. As for the iterative traverse control, the generator repeats to call relevant subroutines, while it selects a subroutine to be called according to the selective traverse control. When the traverse reaches atomic code blocks, the generator modifies codes of adaptable ones with substitution rules, and duplicates codes of rigid ones from template procedures.

### 6.4.3 Merits

Development of various systems based on the *Active*RING model makes it clear that the expansion of small example systems is effective in decreasing burdens of both control engineers and real-time system engineers.

After consistency in a target system definition is checked, a target system is generated so as to reflect configuration and schemata specified in it. Using a generator, control engineers can modify configuration and schemata correctly, without annoying real-time system engineers.

When real-time system engineers describe tags, they may commit errors which lead unexpected behaviors of a generator. Source codes of a generator produced by g_g contribute to identify errors. Suppose a generator fails because of a wrong navigation of arcs in a tag. The source codes of the generator can be scrutinized with a source-level debugger, which indicates a subroutine where the generator fails. Since most subroutines in a generator consist of a few simple lines, it is easy to find a wrong code. The association of one subroutine with one code block facilitates to identify the error in the tag.

In addition to that, the example expansion supports to enhance functions in target systems. Introduction of new mechanisms and devices requires modification of an example system by real-time system engineers. They modify procedures in the example system, and tag modified procedures to produce a new generator. Since an example system is small, modification and tests along with it need far less labors compared with what would be necessary in a large

Table 6.1: Difference between Target System and Example System

| system | requirements | | | | | | | program size(line) | |
|---|---|---|---|---|---|---|---|---|---|
| | schemata | | data items | | objects | | ECA | expanded | unchanged |
| | acq. | prv. | acq. | prv. | acq. | prv. | rules | | |
| example | 3 | 3 | 12 | 12 | 3 | 7 | 12 | 10,483 | 41,338 |
| target | 7 | 367 | 1619 | 4039 | 7 | 2 | 358 | 257,828 | |

target system.

## 6.5 Evaluation in Actual Development

In this section, the effectiveness of the example expansion is evaluated through the development record of a DAS system for a steel mill plant. We show how the example expansion lightens two kinds of burdens: burdens in obtaining the target system, and burdens in obtaining its generator.

### 6.5.1 Efficiency

An example system is expanded according to a target system definition. Table 6.1 summarizes the difference between the target system and the example system. It takes $81man \cdot days$ and $75man \cdot days$ to design the example system and to implement it, respectively. The implementation includes labor for coding, debugging, and testing. As it is known from the table, the example system is converted into the far larger system. The requirement for the target system is specified by a control engineer using the builder. It takes $5man \cdot days$ to specify the requirements and correct errors that the Servish parser points out. It would take $433man \cdot days$ for the same development team to construct the target system from scratch, if it is estimated from the program size. The development time with the example expansion is 2.7 times less than with manual development from scratch. If the development starts at the requirement definition by a control engineer, the labor is 1/87 of the manual development labor.



### 6.5.2 Usefulness of Tool

Let us consider the labor to construct a generator. Table 6.2 shows the time to attach tag to example system procedures and the time to correct errors in tags. Tagging activities are measured for 15 files. There are two kinds of errors. One is a syntax error that g_g can point out. The other is a semantic error which is found out by execution of a generator produced by g_g. Each line in the table shows the time for tagging, the time to correct syntax errors, and the time to correct semantic errors. The last four columns indicates the file sizes without tags and with tags. Since g_g identifies syntax errors, they can be corrected in a short time, as table 6.2 shows. Debugging of semantic errors in tags does not take a long time, either. As mentioned in the previous section, source codes of the generator make it easy to identify semantic errors in tags, because a source level debugger can be used.

Sometimes, example system procedures have to be modified to handle new devices or to add new mechanisms. Table 6.3 shows the time which is necessary to modify tagged procedures. The columns for file sizes compares modified files with original ones. The column for difference indicates the number of different lines. The last column represents the modification is either one taken place only within a code block, or one spreading over several original code blocks. The former is represented by word "intra", while the latter by "inter". As it is easy to understand, it takes more time to reflect modification over multiple code blocks. In one case, 48 lines are modified in the example system. To reflect the modification, two 2 tagged files($o$ and $p$ in table 6.3) are modified. It takes $3man \cdot days$ for the modification of the example system, tests of the example system, and the modification of the tagged files. A target system generated after the modification is different from original one by 5872 lines. From an empirical estimation, the manual modification and tests in a generated system would need 10 times more labor than that taken in the modification. This reveals that the example expansion greatly reduces the time which is necessary for modification and tests.

Table 6.2: Attaching Tags to Expanded Procedures

| file | time (min.) | | | file size | | | |
|---|---|---|---|---|---|---|---|
| | tagging | tag debug | | without tags | | with tags | |
| | | syntax | semantics | (line) | (Byte) | (line) | (Byte) |
| a | 18 | 0 | 0 | 122 | 2424 | 156 | 5021 |
| b | 24 | 0 | 0 | 161 | 3266 | 198 | 5591 |
| c | 117 | 16 | 0 | 662 | 14807 | 732 | 22039 |
| d | 13 | 10 | 0 | 178 | 8516 | 281 | 12250 |
| e | 136 | 11 | 7 | 78 | 2744 | 215 | 12122 |
| f | 81 | 7 | 9 | 90 | 2796 | 238 | 11628 |
| g | 8 | 4 | 0 | 7 | 179 | 38 | 1695 |
| h | 57 | 1 | 2 | 33 | 375 | 129 | 6889 |
| i | 16 | 6 | 2 | 37 | 642 | 89 | 3551 |
| j | 82 | 0 | 12 | 162 | 3797 | 259 | 11352 |
| k | 77 | 4 | 25 | 296 | 8098 | 437 | 20378 |
| l | 99 | 0 | 17 | 903 | 27988 | 1124 | 57943 |
| m | 45 | 1 | 0 | 23 | 687 | 77 | 4869 |

Table 6.3: Modification of Tagged Procedures

| file | time (min.) | | | file size | | | | diff. | inter |
|---|---|---|---|---|---|---|---|---|---|
| | modify | membrane debug | | original | | modified | | | |
| | | syntax | semantics | (line) | (Byte) | (line) | (Byte) | (line) | /intra |
| a | 5 | 0 | 0 | 181 | 5629 | 202 | 6276 | 10 | inter |
| b | 43 | 3 | 21 | 337 | 10095 | 379 | 11789 | 24 | inter |
| h | 9 | 0 | 0 | 109 | 5887 | 129 | 7039 | 20 | intra |
| l | 32 | 0 | 0 | 1120 | 57690 | 1306 | 62865 | 120 | inter |
| o | 13 | 0 | 1 | 268 | 15701 | 320 | 20598 | 38 | intra |
| p | 13 | 0 | 0 | 274 | 16243 | 346 | 21184 | 42 | intra |
| q | 16 | 0 | 0 | 289 | 8935 | 331 | 10201 | 19 | inter |
| r | 120 | 0 | 20 | 510 | 19588 | 559 | 21696 | 22 | inter |
| s | 7 | 0 | 0 | 194 | 9987 | 201 | 10525 | 7 | intra |
| t | 37 | 0 | 0 | 77 | 4869 | 103 | 6388 | 10 | inter |
| u | 13 | 0 | 4 | 129 | 7039 | 128 | 6922 | 14 | intra |
| v | 15 | 0 | 0 | 129 | 7039 | 140 | 7482 | 28 | intra |
| w | 8 | 0 | 0 | 129 | 7039 | 140 | 7482 | 32 | intra |
| x | 13 | 0 | 0 | 129 | 7039 | 140 | 7498 | 32 | intra |



## 6.6 Concentration on Specialities

In this chapter, the example expansion is proposed for the development of a large scale real-time control system without the knowledge on real-time computing.

Using the formal method and the tools to preserve the predictability for maintenance of the timing consistency, a small example is expanded so as to meet requirements for a target system. The example is constructed by real-time system engineers who can implement a mechanism to maintain the timing consistency, while the requirements are specified by control engineers who have expertise in the regulation of a specific plant.

Control engineers benefit from the example expansion because the builder and the generator release them from being conscious of transferring the timing consistency. They can concentrate on achieving stable plant behaviors, which is their speciality. Real-time system engineers have to pay their attention to a small example system. It is when the example must be modified that the development needs them. They only have to work when their expertise is necessary.

The example expansion has been applied to the development of a control system for a steel mill plant. The example expansion has achieved the development which is 2.7 times more efficient than that with other methods. The example expansion also contributes to reducing the development labor of control systems for tunnel ventilation, sewage disposal, and food production.

# Chapter 7

# Concluding Remarks

This thesis discusses the real-time active DAS middleware. From the view points of applications which want to use process values inside various kinds of control machines, the middleware provides a single interface of the control machines. Any application can access process value with the interface. The middleware is also equipped with an active mechanism, with which state changes in a plant are notified to applications. The detection of some state changes needs to be conscious of the time course. The middleware should reflect application requirements including schemata for the data handling and rules for the detection. The middleware can be easily developed according to each application.

The major contributions of the studies explained in the thesis are summarized as follows.

- The middleware integrates service of time dependent data with secured acquisition. The service includes flexible and complicated provision responding to external requests, and the active service according to the data freshness.
- The *Active*RING model is a concise computational model for the middleware, which enables the integration.
- The data modeling the middleware founds can represent the time course, which is indispensable in plant operations.
- The example expansion is proposed to enable the development of large scale control systems without knowledge on real-time computing.

In the thesis, the effectiveness of the real-time active DAS middleware is proved with prototypes and though experiments in actual applications.

Needless to say, the incorporation of semantics specific to a domain amazingly reduces efforts necessary to construct systems to work in the domain. It is the simplicity achieved by the incorporation that makes the real-time active DAS middleware practical and useful. The semantics are reflected in the *Active*RING model which properly relaxes constrains on data and timing behaviors in DAS systems. The simple architecture the model specifies has a suitable abstraction level to represent the tasks in DAS systems. It allows to realize the middleware on COTS real-time operating systems. The *Active*RING model also makes it feasible to develop the formal method and tools for the DAS customization without expertise on real-time computing. The distinction of the two kinds of reactive works contributes to supporting both of the urgent notification and the trend recognition.

The greatest difficulty from which existing commercial DAS systems are suffering lies in the integration of the flexible and complicated service with real-time acquisition. DAS users demand service which needs more complicated computation. They also request more flexible service according to their purposes. The tendency toward the complicated and flexible service shall get stronger, as long as computer technologies continue to prevail in many areas of activities in the manufacturing industry. The real-time active DAS middleware approach gives a promising perspectives to solve the problem. The difficulty for the integration attributes to mixing up real-time tasks with others. Like the discussion in chapter 3 and chapter 4, separation of real-time computation from complicated computation along with flexible responses leads us to a feasible solution using the simplicity of the middleware and the model.

Indeed, there remain problems unsolved in the approach. One of them is that the real-time acquisition ability of a DAS system is likely to reach its limitation because of the enlarged scale of a distributed control system. As it is explained in chapter 2, process values in components of the distributed control system are centralized to a few DAS systems in the current status of the studies. To cope with the scale enlargement of the distributed



control system, a DAS system should be also distributed so that it may correspond to each component. In each component, a small DAS system covers acquisition and detection localized to the component. Even in this case, a supervisory mechanism is necessary to unite series retrieved in distributed DAS systems and to detect state changes associated with interactions among components. Such state changes cannot be found unless process values are monitored over more than one components. The preceding discussion implies the necessity of two kinds of DAS systems: local DAS systems and central DAS systems. The local DAS systems would execute simple tasks in a small hardware and software platform for each component in the distributed control system. Working as a supervisor, the central DAS systems should synthesize retrieval results and notification messages from distributed DAS systems. The clock synchronization is indispensable in each of the DAS systems. It is also worthy of attention that developers of such DAS system should be free from being conscious of the distribution and the centralization. Investigation activities have been already started to scrutinize requisites to solve the problem.

Although it is obvious in the above discussion, the real-time active DAS middleware and the *Active*RING model on which the middleware founds have not attained the sound solution to solve all problems that DAS systems have suffered from or may be potentially faced with. Nevertheless, the simplicity of the middleware and the model is expected to be strong enough to solve the problems on COTS hardware/software platforms.

# Acknowledgment

The author would like to express a sincere appreciation to his supervisor, Professor Yahiko Kambayashi of Kyoto University for his continuous encouragements, comprehensive suggestion and constructive criticisms during the course of the studies.

The author wishes to thank deeply Associate Professor Hiroyuki Tarumi of Kyoto University for his stimulating advice and valuable discussion.

The author is grateful to Professor Krithi Ramamritham of University of Massachusetts for his comments and discussion of insights.

The author expresses his gratitude to Dr.Ichiro Nakahori and Dr.Morikazu Takegaki, who gave the author the chance and the supports of the studies. The author also appreciates Dr.Satoshi Horiike. Without his help, the author could not accomplish the customization tools.

The thanks of the author due to the members of Mitubishi Electric Cooperation, including George Ido, Hideyuki Takada, Minoru Sugitani, Seiji Shitaoka, Masao Ijiri, Yoshitomo Asano, Ichiro Mizunuma, Kouji Kikkawa, Kouichi Nakagawa, Hideji Ohnishi, and Kenji Nishimura for their contributions and cooperation on the development projects related to the real-time active DAS middleware.

# References


[1] J.F.Allen, Towards a General Theory of Action and Time, *Artificial Intelligence*, Vol.23, No.2, pp.123-154, 1984

[2] P.A.Bernstein, Middleware, *Comm. ACM*, Vol.39, No.2, pp.86-98, 1996.

[3] S.Boyer, *SCADA: Supervisory control and data acquisition*, Instrument Society of America, 1993.

[4] A.Cornelio, and S.B.Navathe Using Active Database Techniques for Real-Time Engineering Applications *Proc. 9th ICDE*, 1993.

[5] G.F.Coulouris, J.Dollimore, *Distributed Systems - Concept and Design*, Addison-Wesley Pub., 1988

[6] C.D.Date, *An Introduction to Database Systems*, Fifith Edition, Volume.II, Chapter.5, pp.181-240, Addison Wesley, 1990.

[7] R.Dechter, Temporal Constraint Network, *Artificial Intelligence* No.49. pp.61-95,(1991).

[8] D.del Val, and A.Viña, Applying RMA to Improve a High-Speed, Real-Time Data Acquisition System, *Proc.of 15th Real-Time System Symposium*, 1994.

[9] R.Elmasri, G.T.Wuu, and Y.Kim, The Time INDEX: An Access Structure for Temporal Data, *Proc.of 16th VLDB*, pp.1-12, 1990

[10] M.S.Fox, An Organizational View of Distributed Systems, *IEEE Trans. on System, Man, and Cybernetics*, Vol.SMC-11, No.1, 1981.

[11] K.B.Gallagher, and R.Lyle, Using Program Slicing in Software Maintenance *IEEE Trans. on Software Engineering*, Vol.17, No.8, pp.751-761, 1991.

[12] L.Gasser, C.Braganza, and N Herman, MACE: A Flexible Testbed for Distributed AI Research. In M.N.Huhns (ed.), *Distributed Artificial Intelligence*, pp.119-152, Morgan Kaufmann Publishers, 1987

[13] R.Gerber, and S.Hong, Slicing Real-Time Programs for Enhanced Schedulability *ACM Trans. on Programming Languages and Systems*, Vol.19, No.3, pp.525-555, 1997.

[14] Marc H.Graham, Issues in Real-Time Data Management, *J. of Real-Time Systems*, Vol.4, 1992

[15] J.R.Haritsa, M.Livny, and M.J.Carey, Earliest Deadline Scheduling for Real-Time Database Systems, *Proc.of 12th Real-Time System Symposium*, 1991.

[16] C.Hewitt, Viewing Control Structures as Patterns of Passing Messages, *Artificial Intelligence*, Vol.8, pp.323-364, 1977.

[17] D.Hongm, T.Johnson, and S.Chakravarthy, Real-Time Transaction Scheduling: A Const Conscious Approach, *Proc.of the ACM SIGMOD Intenational Conference on Management of Data*, pp.197-206, 1993.

[18] M.N.Huhns(ed), *Distributed Artificial Intelligence*, Morgan Kaufmann Publishers(1987).

[19] Intellution home page, *http://www.intellution.com/*

[20] Y.Ishikawa, H.Tokuda, and C.W.Mercer, An Object-Oriented Real-Time Programming Language, *Computer*, Vol.24, No.10, pp.67-73, 1991

[21] K.Katori, S.Kazama, S.Abe, A.Miyajiri, and K.Yamanaka, Present and Future Trends in Industrial Computer Systems, *Mitsubishi Denki Giho*, Vol.70, No.7, pp.2-6, 1996. (in Japanese)



[22] Y.Kim, M.R.Lehr, D.W.George, and S.H.Son, A Database Server for Distributed Real-Time Systems: Issues and Experiences, *Proc.of 2nd Internaltional Workshop on Parallel Distributed Real-Time Systems*, 1994.

[23] Y.Kim, and S.H.Son, Supporting predictability in Real-time Database Systems, *Proc.of 2nd Real-Time Application and Technology Symposium*, 1996.

[24] M.H.Klein, T.Ralya, B.Pollak, R.Obenza, and M.G.Harbour, *A Practitioner's Handbook for Real-Time Analysis*, Kluwer Academic Publishers, 1993.

[25] T.-W.Kuo, and A.K.Mok, Incremental Reconfiguration and Load Adjustment in Adaptive Real-Time Systems, *IEEE Trans.on Computers*, Vol.46, No.12, pp.1313-1325, 1996.

[26] C.L.Liu and J.W.Layland, Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, *J. ACM*, Vol.20, No.1, pp.46-61, 1973

[27] M.Maekawa, *Operating System*, Chapter.3, pp.49-122, Iwanami, 1988. (in Japanese)

[28] R.Maiocci, and B.Pernici, Temporal Data Management Systems: A Comparative View, *IEEE Trans. on Knowledge and Data Engineering*, Vol.3, No.4, 1991.

[29] D.R.McCarthy and U.Dayal, The Architecture of An Active Database management Systems, *Proc.of ACM SIGMOD*, 1989.

[30] D.McDermott, A Temporal Logic Reasoning About Process and Plan, *Cognitive Science*, No.6, pp.101-155, 1982.

[31] D.Meyer and C.Cannon, *Buliding A Better Data Warehouse*, Prentice Hall, 1998

[32] J.Muratore, T.Heindel, T.Murphy, A.Rasmussen, and Z.McFarland, Acquisition at Mission Control, *Comm. ACM*, Vol.33, No.12, 1990.

[33] H.Narazaki, Expert Systems Application to Flatness Control of Aluminum Foil Rolling, *Trans. of the Institute of Systems, Control, and Information Engineers*, Vol.3, No.11, pp.358-371, 1990 (in Japanese).

[34] D.Niehaus, J.A.Stankovic, and K.Ramamritham, A Real-Time System Description Language, *Proc.of 1st Real-Time Application and Technology Symposium*, 1995

[35] Object Automation home page, *http://www.oa.com/*

[36] T.Ozkul, *Data Acquisition and Process Control using Personal Computers*, Marcel Dekker, Inc., 1996

[37] Gultekin Özsoyoğlu and Richard T.Snodgrass, Temporal and Real-Time Database: A Survey, *IEEE Trans. on Knowledge and Data Engineering*, Vol.7, No.4, 1995

[38] C.Peng, and K.J.Lin, A Semantic-Based Concurrency Control Protocol for Real-Time Transactions, *Proc.of 2nd Real-Time Application and Technology Symposium*, 1996.

[39] R.Rajkumar, *Synchronization in Real-Time Systems, A priority Inheritance Approach*, Kluwer Academic Publishers, 1991

[40] K.Ramamritham, Real-Time Database, *International Journal of Distributed and Parallel Database*, Vol.1, No.2, March 1993

[41] K.Ramamritham, R.Sivasankaran, J.A.Stankovic, D.T.Towsley, and M.Xiong, Integrating Temporal, Real-Time, and Active Databases, *SIGMOD Record*, Vol.25, No.1, March 1996

[42] T.Risch, Monitoring Database Object, *Proc. of 15th VLDB*, pp.445-453, 1993

[43] SAP home page, *http://www.sap-ag.de/*

[44] J.D.Schoeffler, Distributed Computer Systems for Industrial Process Control, *IEEE Computer*, Vol.17, No.2, 1984

[45] P.Seshadri, M.Livny, and R.Ramakrishnan, The Design and Implementation of a Sequence Database System, *Proc.of 22nd VLDB*, 1996.

[46] L.Sha and S.S.Sathaye, Architectual Support for Real-Time Computing using Generalized Rate Monotonic Theory, *J.of The Society of Instrument and Control Engineering*, Vol.31. No.7, 1992.



[47] H.Shimakawa, and S.Ikebata The Structured Discription of The Knowledge in The Situation Recognition, *Proc. of Internaltional Workshop of AI for Industrial Application*, 1988

[48] H.Shimakawa, Y.Sugitani, K.Kikkawa, and T.Watanabe, Agents to Guide Operators with Recognition of Time Series, *Proc.of 16th COMPSAC*, 1992.

[49] H.Shimakawa and K.Kikkawa, Trend Recongnition with Time Series Database. *Proc.of 2nd Far-East Workshop on Future Database System*, pp.373-382, 1993.

[50] H.Shimakawa, H.Ohnishi, I.Mizunuma, and M.Takegaki, Acquisition and Service of Temporal Data for Real-Time Plant Monitoring, *Proc.of 14th Real-Time System Symposium*, 1993.

[51] H.Shimakawa, I.Mizunuma, and M.Takegaki, Real-Time Data Server to Acquire and Serve Temporal Data in Plants, *Trans. of the Institute of Electronics, Information, and Communication Engineeres*, Vol.J78-D-I, No.8, pp.798-806, 1995. (in Japanese)

[52] H.Shimakawa, G.Ido, H.Takada, and M.Takegaki, Active Transactions Integrated with Real-Time Transactions According to Data Freshness, *Proc. of 3rd Real-Time Technology and Application Symposium*, 1997.

[53] Y.Shoham, Time for Action: On the relation between time, knowledge, and action, *Proc. of the 11th International Joint Conference on AI*, pp.954-959, (1989)

[54] R.M.Sivasankaran, J.A.Stankovic, D.Towsley, B.Purimetla, K.Ramamritham, Priority Assignment in Real-Time Active Databases, *The VLDB Journal*, Vol.5, No.1, January, 1996.

[55] X.C.Song and J.W.S.Liu, Maintaining Temporal Consistency: Pessimistic vs. Optimistic Concurrency Control, *IEEE Trans. on Knowledge and Data Engineering*, Vol.7, No.5, 1995

[56] B.Sprunt, L.Sha, and J.Lehoczky, Aperiodic Task Scheduling for Hard-Real-Time Systems, *Journal of Real-Time Systems*, Vol.1, No.1, 1989.

[57] J.A.Stankovic and K.Ramamritham, The SPRING Kernel: A New Paradigm for Real-Time Systems, *IEEE Software*, Vol.8, No.3, pp.62-72, 1991

[58] D.B.Stewart, R.A.Volpe, and P.K.Khosla, Design of Dynamically Reconfigurable Real-Time Software using Port-Based Objects, *IEEE Trans. on Software Engineering*, Vol.23, No.12, pp.759-776, 1997

[59] O.Takada, Conputer Network in Process Control, *Journal of Institute of System, Control, and Information*, Vol.38, No.10, 1994. (in Japanese)

[60] A.U.Tansel, J.Clifford, S.Gadia, S.Jajodia, A.Segev, and R.Snodgrass, *Temporal Databases, Theory, Design, and Inmplementation*, Benjamin/Cummings Pub., 1993.

[61] J.M.Voas, The Challenges of Using COTS Software in Component-Based Development, *Computer*, Vol.31, No.6, pp.44-52, 1998.

[62] J.Widom and S.Ceri, ed., *Active Database Systems, Triggers and Rules for Advanced Database Processing*, Morgan Kaufmann Pub., 1996.

[63] Wonderware home page, *http://www.wonderware.com/*.

[64] L.Zhou, E.A.Rundensteiner, and K.G.Shin, Schema Evolution of an Object-Oriented Real-Time Database System for Manufacturing Automation, *IEEE Trans. on Knowledge and Data Engineering* Vol.9, No.6, pp.956-977, 1997



# List of Publications and Technical Reports

## Major Publications

[1] H.Shimakawa and S.Ikebata, Classification of Concepts and Knowledge on Problem Solving in Situation Recognition, *Proc. of IPSJ Symposium on Frameworks of Artificial Intelligence Systems*, Tokyo, June, 1987 (in Japanese)

[2] H.Shimakawa, and S.Ikebata, The Structured Description of the Knowledge in the Situation Recognition, *Proc. of International Workshop of AI for Industrial Application*, Hitachi, Feb., 1988

[3] H.Shimakawa, and S.Ikebata, Distributive Watch Using Situation Recognition, *Proc. of the First International Workshop on AI, Simulation, and Planning in High Autonomy Systems*, pp.51-57, Tucson, March, 1991

[4] H.Shimakawa, and K.Kikkawa, Trend Recognition with Time Series Database, *Proc. of the 2nd Far-East Workshop on Future Database System*, pp.373-382, Kyoto, April, 1992

[5] H.Shimakawa, Y.Sugitani, K.Kikkawa, and T.Watanabe, Agents to Guide Operators with Recognition of Time Series, *Proc. of the 16th Computer Software and Application Conference*, pp.312-319, Chicago, Sept., 1992

[6] H.Shimakawa, H.Ohnishi, I.Mizunuma, and M.Takegaki, Acquisition and Service of Temporal Data for Real-Time Plant Monitoring, *Proc. of the 14th IEEE Real-Time Systems Symposium*, pp.112-119, Raleigh-Durham, Dec., 1993

[7] H.Shimakawa, I.Mizunuma, and M.Takegaki, Real-Time Data Server to Acquire and Serve Temporal Data in Plants, *Trans. on the Institute of Electronics, Information and Communication Engineers*, Vol.J78-D-I, No.8, pp.798-806, Aug., 1995 (in Japanese)

[8] H.Shimakawa, Current Status and Problems in Real-Time Supervisory Control of Plants, *Journal of Institute of Systems, Control and Information Engineers*, Vol.41, No.5, pp.172-178, May, 1997 (in Japanese)

[9] H.Shimakawa, G.Ido, H.Takada, and M.Takegaki, Active Transactions Integrated with Real-Time Transactions According to Data Freshness, *Proc. of the Third IEEE Real-Time Technology and Applications Symposium*, pp.49-59, Montreal, June, 1997

[10] H.Takada, H.Shimakawa, Y.Asano, J.Ido, and M.Takegaki, A Database Management System Based on an Object Model with Temporal Validity for Plant Monitoring Applications, *Trans. of the Institute of Electronics, Information and Communication Engineers*, Vol.J79-D-I, No.10, pp.853-862, Oct., 1996 (in Japanese)

[11] H.Takada, H.Shimakawa, Y.Asano, and M.Takegaki, Production Information Management for Batch Manufacturing Plants Based on ECA Mechanism and View Generation, *Proc. of International Workshop on Databases : Active and Real-Time*, pp.92-97, Rockville, Nov., 1996.

[12] H.Takada, H.Shimakawa, and M.Takegaki, A Batch Manufacturing Information Management System Based on Multidimensional View Generation and Active Mechanism, *Proc. of International Symposium on Cooperative Database Systems for Advanced Applications*, pp.321-328, Kyoto, Dec., 1996

[13] H.Takada, H.Shimakawa, Y.Asano, and M.Takegaki, A Batch Manufacturing Information Management System Based on Active Mechanism and Multidimensional View Generation, *Trans. of the Institute of Systems, Control and Information Engineers*, Vol.10, No.11, pp.575-584, Nov., 1997 (in Japanese)

[14] I.Mizunuma. H.Shimakawa, and M.Takegaki, Real-Time Middleware Based on Rate Monotonic Theory, *Proc. of the 2nd IEEE Workshop on Real-Time Applications*, pp.58-62, Washington, July, 1994

[15] I.Mizunuma, H.Shimakawa, and M.Takegaki, Real-Time Middleware for Reliable Industrial Applications, Denshi Tokyo, No.33, pp.132-137, Dec., 1994

[16] I.Mizunuma, H.Shimakawa, and M.Takegaki, Real-Time Middleware for Industrial Applications, *Proc. of 1st International Workshop on Real-Time Computing Systems*, pp.101-105, Seoul, Dec., 1994

[17] I.Mizunuma, H.Shimakawa, H.Kanamaru and M.Takegaki, A Design Method for Stand-by-Redundant Systems Guaranteeing Timing Constraints, *Trans. of the Institute of Electronics, Information and Communication Engineers*, Vol.J78-D-I, No.8, pp.756-769, Aug., 1995 (in Japanese)

## Technical Reports

[1] H.Shimakawa, G.Ido, Y.Asano, and M.Takegaki, Introduction of ECA Mechanism into Real-Time Data Server, *Technical Report of IEICE.*, CPSY94-112 March, 1995 (in Japanese)

[2] H.Takada, H.Shimakawa, Y.Asano, and M.Takegaki, Utilization of the Active Mechanism and View Support Functions for Manufacturing Management Databases, *Technical Report of IEICE.*, DE95-40 July, 1995 (in Japanese)

[3] J.Ido, and H.Shimakawa, Real-Time Data Server to Acquire and Serve Periodic/Aperiodic Data in Plant, *Technical Report of IEICE.*, DE95-46 July, 1995 (in Japanese)

[4] H.Takada, H.Shimakawa, Y.Asano, and M.Takegaki, A Temporal Object Model with Valid Interval Based Temporal Validity and Its Application, *Technical Report of IEICE.*, DE96-35 July, 1996 (in Japanese)

[5] H.Shimakawa, G.Ido, and M.Takegaki, RTDS: A Data Acquisition System Satisfying Timing and Temporal Constraints, *Technical Report of IEICE.*, DE96-36 July, 1996

(in Japanese)

[6] J.Ido, H.Shimakawa, and M.Takegaki, Evaluation of Real-Time Data Server, *Technical Report of IEICE.*, DE96-37 July, 1996 (in Japanese)

*IPSJ*: Information Processing Society of Japan

*IEICE*: Institute of Electronics, Information and Communication Engineers